Fermilab

GU0001

UNIX at Fermilab

Release 3.0

November 26, 1997

Computing Division
Fermi National Accelerator Laboratory

Compiled by Anne Heavey

ABSTRACT

The *UNIX at Fermilab* manual has several goals: to help you get started as quickly and painlessly as possible in the UNIX environment here at Fermilab, to make you aware of the UNIX resources available at Fermilab, and to help you use UNIX easily and efficiently, whatever your particular needs.

We discuss the most commonly used UNIX concepts, commands and tools, and provide enough information to allow you to execute commands and perform tasks typically required by the Fermilab user community. We refer you to more specialized Fermilab documents as needed, and to commercially available sources of information for details on commands and features that will allow you to exploit the more sophisticated features of the UNIX operating system not covered in this document.

Revision Record

August 1990	Original Draft
March 1996	Total overhaul; then several minor revisions through September 1996
November 1997	Significant changes relative to 9/30/96 release (text changes *within* sections are not noted here). Chapter and section numbers correspond to new release:

Ch 1 (Introduction): new FNALU section (1.5)

Ch 2 (Getting Started): moved AFS information into AFS chapter (7)

Ch 3 (Information Resources): new Helpdesk section (3.5)

Ch 6 (File System): new tar section (6.3.7); moved AFS information into AFS chapter (7)

Ch 7 (AFS): new chapter

Ch 9 (Work Environment): added section on X-terminal support (9.5)

Ch 10 (UPS Products): new chapter (replacement)

Ch 11 (Editors): new information for the editor xemacs (11.3.2)

Ch 12 (Mail): new "Notice of Upcoming Changes"

Ch 14 (Batch): new chapter

Ch 15 (Tapes): condensed OCS coverage; added sections for FTT (15.4) and FMB (15.5)

Ch 16 (Sofware Dev): new section on the debugger GDB (16.11.3)

Ch 18 (Code Mgmt): condensed the UCM section (18.2)

Removed appendix on FUE Conformance Levels

Appx B (UPS Overview): new appendix

Appx F (mh and exmh Custom): new section on auto reply to mail (F.3)

Acknowledgments

In 1995-1996, the Fermilab Computing Division's VMS Migration Task Force was responsible for updating *UNIX at Fermilab* (from the June 1991 release) in time for the mass migration from FNALV. That task force included: Pushpa Bhat, Joel Butler, Lauri Loebel Carpenter, Mike Diesburg, Dave Fagan, Ken Fidler, Steve Hanson, Rob Harris, Joy Hathaway, Anne Heavey, Art Kreymer, Paul Lebrun, Frank Nagy, Judith Nicholls, Dane Skow, Mike Stolz, Roy Thatcher, Al Thomas, Matt Wicks, and Steve Wolbers.

Other contributors/reviewers included Liz Buckley-Geer, Chuck DeBaun, Lynn Garren, Randy Herber, Peter Kasper, Marc Mengel, Clyde Moseberry, Gene Oleynik, Ruth Pordes, Marilyn Schweitzer, Jonathan Streets, and Margaret Votava.

The *CERN UNIX User Guide*, after having lifted information from the early release of *UNIX at Fermilab*, was in turn used to provide much of that information back to us appropriately updated! Naturally, we lifted additional useful information from CERN's well-constructed manual. The CERN document *Shell Choice, A shell comparison* was also used.

This 1997 revision of *UNIX at Fermilab* includes contributions from many of the people listed above. Additional contributors/reviewers include Eileen Berman, Jim Fromm, Lisa Giacchetti, Jeff Kallenbach, John Marraffino, Laura Mengel, Liz Sexton-Kennedy, and Alan Wehmann.

We'd like to thank the readers who have taken the time to send us suggestions about the manual.

Table of Contents

Chapter 1: Introduction

This chapter provides an introduction to the manual itself as well as to UNIX and its implementations at Fermilab. In particular, it covers:

- the purpose and intended audience of *UNIX at Fermilab*
- where to send comments and questions about the manual
- where to obtain additional copies of the manual
- the typeface conventions and symbols used throughout the manual
- a summary of the contents of all the chapters and appendices in the manual
- a very brief history of the UNIX operating system and a discussion of its advantages
- an introduction to the *Fermi UNIX Environment*, which is installed on most UNIX systems at Fermilab
- a listing of the supported UNIX operating systems
- an introduction to FNALU, the central Fermilab UNIX cluster
- information on locating on-line Fermilab documents
- a listing of UNIX references

1.1 The *UNIX at Fermilab* Manual

This manual covers basic UNIX concepts and operations, and thus is especially geared to users who are new to UNIX. It also documents the UNIX environment and file systems commonly used at Fermilab, and thus is a handy reference for all users of Fermilab UNIX systems.

> There are many different UNIX systems installed at Fermilab, with differing levels of conformance to Fermilab standards. Therefore, some of the information contained in this manual may not be valid for your system. You will very likely need to obtain supplementary documentation and information specific to your system.
>
> In particular, CDF and D0 have created documentation for their systems and made it available on the World Wide Web (this utility is described in section 3.2.1). You can access this information at the following locations:
>
> **CDF**
>
> `http://www-cdf.fnal.gov/offline/cdfsga/cdfsga.html`
>
> **D0**
>
> `http://d0wop.fnal.gov/d0unix/d0unix.html`

The purpose of this manual is to help you get started as quickly and painlessly as possible in the UNIX environment here at Fermilab, as well as to help you use UNIX easily and efficiently, whatever your particular needs. We discuss the most commonly used UNIX concepts, commands and tools, and provide enough information to allow you to execute commands and perform tasks typically required by the Fermilab user community. We refer you to commercially available sources of information for details on commands and features that will allow you to exploit the more sophisticated features of the UNIX operating system not covered in this document. We also describe the Fermi UNIX Environment, which includes Fermilab-specific programs, libraries, tools, and usages, and occasionally refer you to other Fermilab documents.

We try to describe here a rather "standard UNIX" and recommend that you try not to use the "value-added" features that the individual vendors provide which may cause you trouble in conversion to another platform. Commands and features for both families of UNIX command interpreters (discussed in Chapter 4), C shell and Bourne shell, are included when they differ. Since some of the systems at Fermilab will have the AFS file system installed, we address some issues concerning AFS in Chapter 7.

It is *not* the purpose of this manual to teach you UNIX or to give you a complete description of UNIX. For that purpose we recommend that you obtain and read one of many good books on UNIX. See section 1.6 for some suggestions. This manual is also not intended as a system administration reference.

For those readers who wish to convert from VMS to UNIX quickly without learning about UNIX ahead of time, see Appendix A for the bare minimum needed to accomplish this task!

UNIX at Fermilab is bound to contain some errors, however we endeavor to minimize the error count! We encourage all the readers of this manual to report back to us:

- errors or inconsistencies that we have overlooked
- any parts of the manual that are confusing or unhelpful -- please offer *constructive* suggestions!
- other topics to include (keeping in mind the purpose of the manual)
- tricks that other Fermilab users might find helpful

Send your comments via email to *cdlibrary@fnal.gov.*

Copies of *UNIX at Fermilab*, document number GU0001, can be obtained from the following sources:

WWW[1]	`http://www.fnal.gov/docs/UNIX/unix_at_fermilab` You can reach this from the Fermilab Computing Division home page, under *UNIX Resources* or via the document database (see section 1.6.1). Both PostScript and HTML versions are available.
Paper Copies	Wilson Hall, 8th floor, NE (outside the former Computing Division library)
AFS	PostScript files are available under `/afs/fnal/files/docs/UNIX/unix_at_fermilab/ps/rev1997/`

1. See section 3.2.1 for information on the World Wide Web (WWW).

1.1.1 Conventions Used in this Manual

The following notational conventions are used in this manual:

bold	Used for shells (e.g., **csh**) and product names.
italic	Used to emphasize a word or concept in the text; many terms found in the index are italicized in the text. Also used to indicate variables and reference book titles.
`typewriter`	Used for filenames, path names, contents of files, output of commands.
sans-serif	Used to indicate keys (on keyboard) and "buttons" on graphical applications.
`typewriter-bold`	In text, used to indicate commands and prompts. In command formats, indicates what the user types "as is".
bold-italic	In command formats, indicates variables for which the user must make context-specific substitutions.
<Ctrl-***char***>	Indicates a control character. To enter a control character, hold down the control key (labeled Ctrl, probably) while pressing the key specified by ***char***.
[ ]	In command formats, indicates **optional** command arguments and options.
|	When shown in between brackets (**`[x|y|z]`**) in a command line, separates a series of options from which one must be chosen. In UNIX command formats, used to "pipe" output of preceding command to the following one (see section 5.4.3).
' ... '	Single vertical quotes indicate apostrophes in commands.
` ... `	Single backquotes in UNIX commands have a special meaning (use output of string in place of string itself). Don't confuse them with apostrophes.
. . .	Means that a repetition of the preceding parameter or argument is allowed.
%	Prompt for C shell family commands (**%** is also used throughout this document when a command works for both shell families).
$	Prompt for Bourne shell family commands; also standard UNIX prefix for environment variables (e.g., *$VAR* means "the value to which *VAR* is set").
{ }	In environment variables, paths, files and other text strings, indicates strings for which the user must make context-specific substitutions. For example, *${PRODUCT}_DIR* is *$WWW_DIR* for the product **www**, and `{username}@{node}` refers to a string like `joe@fsui02`.

All command examples are followed by an implicit carriage return key.

The following symbols are used throughout *UNIX at Fermilab* to draw your attention to specific items in the text:

A UNIX "bomb"; this refers to something important you need to know about UNIX in order to avoid a pitfall.

A VMS "bomb"; this is used to point out common pitfalls of VMS users.

This symbol indicates information for AFS systems (see Chapter 7).

This symbol is intended to draw your attention to a useful hint.

This is a reminder.

We refer you to other documentation sources for further information where you see this symbol.

1.1.2 Summary of Chapters and Appendices

The organization of this manual reflects our compromise between providing a tutorial and a reference manual. You may find it useful to skip around a little rather than reading straight through.

We recommend that you get a UNIX account and begin to work with the system as you read.

Chapters:

2: Getting Started on a UNIX System

This chapter describes the login and logout procedures, and discusses some basic UNIX features including the prompt, special keys, and special characters. We present a brief discussion of the file systems you are likely to encounter at Fermilab, and how to change your password according to your file system and installation.

3: Information Resources

This chapter introduces you to the information facilities available from UNIX. Standard UNIX on-line help is available via the man pages. We discuss the World Wide Web and newsgroups, which are very rich sources of information on a virtually unlimited set of topics. A few utilities that allow you to get information about vendor products, other users, and the Fermilab computing systems are also covered. Finally, we include instructions for communicating with the Fermilab Helpdesk.

4: Shells

This chapter discusses the concept of a UNIX shell, and how to manipulate shells. It includes information on the available and recommended shells and their features. The concept of a shell as an interpretive programming language is introduced.

5: Important UNIX Concepts

This chapter introduces you to the UNIX command structure, and to many important commands and concepts. The features introduced in this chapter constitute the core of the UNIX operating system, and many of these tools are quite powerful and flexible. Some of the features are shell-specific, and we provide the distinctions where necessary.

6: The UNIX File System

The UNIX file system has a hierarchical or tree-like structure with the directory called *root* (**/**) as its source. The system is essentially composed of *files* and *directories*. In this chapter we describe techniques for manipulating files and directories, and commands designed to provide information about them.

7: The AFS File System

Fermilab is using the AFS (Andrew File System) as a distributed file service model, and it is installed on several machines on site in a production environment, including the FNALU cluster. This chapter discusses the basic concepts of AFS and provides information on the commands used to manage your files and directories in the AFS environment.

8: Printing

This chapter covers the standard FUE print utility **flpr**, as well as filter programs and techniques available for formatting the output prior to printing. **flpr** is the Fermi implementation of the standard UNIX **lpr** utility. Most software applications supplied by the Computing Division use **flpr** as a default.

9: Working Environment

This chapter describes the methods used to set up your working environment in UNIX. Some of these are standard UNIX (e.g., shell and environment variables), and some are provided and/or customized by FUE (e.g., the login scripts).

10: Accessing Software Products

In this chapter you will learn how to get information about the software products that are provided and supported by the Computing Division. We describe how to access products already installed on your system, and how to obtain a product from the Fermilab `KITS` area and install it on your system.

A notice regarding upcoming changes is included.

11: Editors

Several text editors are available at Fermilab. In this chapter we present our view of the advantages and disadvantages of the available editors, and we provide some basic information on the setup and use of each one. You will learn how to invoke each editor, and how to create, edit, and save a file in each one using a small subset of commands and features. We include only minimal usage information for the VMS-style editors.

12: UNIX Mail Systems

This chapter describes how mail forwarding is managed at Fermilab and discusses the UNIX mail handlers that are currently available and supported.

A notice regarding upcoming changes is included.

13: Connecting to Remote Systems

Several utilities are available to enable you to transfer files between systems, to log into other systems on which you have an account, and to execute commands remotely. These features are described in this chapter.

14: Batch Processing Environment

In this chapter we provide introductory information on **LSF** (Load Sharing Facility), the standard batch processing system at Fermilab, and on **fbatch**, the locally-written interface to **LSF**. We also list the related software components that can be used with **LSF/fbatch**.

15: Tape Handling

In this chapter we discuss the principal tape handling software and facilities available at Fermilab. Start-up information for running and monitoring **OCS** tape mounts is provided, and the **OCS** X interface is introduced. Several tape I/O packages are briefly described.

16: Software Development

This chapter gives an introduction to UNIX software development tools in common use at Fermilab, providing information on:

- Supported languages
- Compiling and linking in C, C++ and FORTRAN
- Debugging

We do not include a discussion of general programming here, but rather, aspects of software development particular to UNIX.

17: The make Utility

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. Although **make** can be used in a wide variety of applications, in this chapter we concentrate on its use in the area of software development. We describe how to define relationships between source, object, library and executable files for use by **make**, and how to invoke **make** in its simplest and slightly more complex forms.

18: Code Management

This chapter introduces the recommended code management solution for UNIX, **CVS** (Concurrent Versions System). We also introduce an alternative that is currently being used by a couple of Fermilab experiments, **UCM** (UNIX Code Management). Both packages use **RCS** (Revision Control System) as the underlying protocol. We provide basic information only, and refer you to the complete manuals for these utilities for detailed information.

Appendices:

A: VMS Migration for the Impatient

So, you've decided you're ready to convert (or you've run up against a deadline!), but you don't know the first thing about UNIX. Here's enough information to get you moved over. You can use the rest of the manual to learn about UNIX afterwards.

B: UNIX Product Support (UPS) Overview

In this appendix we discuss the Fermilab product support structure, **UPS**. We recommend that you read and understand this material before performing any of the tasks described in Chapter 10.

A notice regarding upcoming changes is included.

C: Fermi Login Files

This appendix contains file listings of the FUE-customized default login files (the "Fermi Files") used to set up your UNIX environment. If you are on a FUE-compliant system, you are supplied with a copy of each file in your home directory (except for the `fermi.*` and `setup.*` files which are executed directly from `/usr/local/etc`). The files are reprinted here in their entirety except for the copyright disclaimers.

D: awk's Programming Model

This appendix is adapted from a section of the same name in the book *sed & awk*, published by O'Reilly & Associates. It describes the generic structure and organization of an **awk** program.

E: VMS to UNIX Command Reference

This appendix is intended as a convenient UNIX command reference for the migrating VMS user. The UNIX equivalents of commonly used VMS commands are listed in tabular format. No information on syntax, options, or arguments is presented here, however some of the listed commands are described elsewhere in *UNIX at Fermilab.*

F: mh and exmh Customization

This appendix contains information for further customizing the **mh** and/or **exmh** mail readers. Given the multitude of customizable features in these mail readers, we cannot provide you with a comprehensive treatment of the subject here. The information presented in Chapter 12 and this appendix should be sufficient for most users.

G: mh Command Reference

This appendix provides an alphabetical reference to the subset of **mh** commands discussed in Chapter 12.

H: Mail Conversion from VMS

This appendix is intended to guide you through a mail conversion process from VMS to the UNIX **MH** (Message Handling) system. Two options for conversion are presented.

I: Programming Examples

This appendix contains examples of programs that illustrate information presented in Chapters 16 and 17.

1.2 About UNIX

1.2.1 A Brief History

The UNIX operating system was developed at AT&T Bell Laboratories in Murray Hill, New Jersey in the late 1960's. Universities and colleges have played a major role in popularizing UNIX. The Computer Science Department at the University of California at Berkeley made so many popular changes to it that one of the two most popular versions in use today is named the Berkeley Software Distribution (BSD) of the UNIX system. The other major version is AT&T's UNIX System V. Many implementations incorporate features of both systems.

1.2.2 A Word About Features and Components

Because UNIX was originally designed by programmers to support their own projects, one of its strongest points is that it provides an excellent software development environment. UNIX has a large set of powerful utility programs and tools that allow users to easily build systems and applications. It also has several command interpreters, called *shells* that can also be used as high-level programming languages. Keep an open mind to the powerful features of UNIX that may be quite different from systems you are familiar with. Some of the important concepts are introduced in Section 5.4 (e.g., pipes and filters, and device-independence).

Bear in mind that there are many UNIX utilities not described in this guide.

The UNIX operating system has four basic components:

- The *kernel* constitutes the nucleus of the operating system and coordinates the internals such as allocating system resources.
- The *file system*, which is part of the kernel, controls the storage and access of data. It is similar to the VMS file system in that the structure is hierarchical.
- *Commands* are programs that you request the computer to execute.
- Programs (commands) called *shells* serve as command interpreters. They act as links between you and the computer, interpreting and executing commands. They are also high-level interpretive programming languages.

Although there are many implementations of UNIX, there are two families of shells. The Bourne shell family and the Berkeley/C shell family. The shell families and the individual shells are discussed in more detail in Chapter 4.

1.2.3 The Advantages of UNIX

New Technologies

UNIX allows Fermilab to achieve better alignment with the mainstream of computing. The "developmental momentum" is with UNIX and personal computers, and it is primarily for these environments that the new technologies, software products, and software methods are emerging. Other HEP laboratories have also recognized these trends and are moving in this direction.

Multi-Platform Support

There are many UNIX platforms, and while there are certainly variations among them, the differences tend to be small. This allows the Computing Division to more effectively support platforms supplied by many vendors, thus eliminating the link to a single vendor whose pricing policies and technology directions are outside of our control.

Availability of Popular Applications

The general administrative and much of the technical computing load has migrated to MACs and PCs due to the availability of commercial applications. There is also a growing demand from HEP researchers for these applications. The ability to use personal computers as X-terminals offers a real possibility of providing a single platform that allows access to popular applications for word-processing, preparation of presentations, charting, statistical analysis, etc., as well as access to powerful UNIX systems for data analysis and program development.

Choice of Machine and Platform

You first need to determine if you are going to purchase your own desktop UNIX system, get an account on a workgroup UNIX system, or get an account on the central Fermilab UNIX system, FNALU. At this point, you may or may not have a choice of platform; for instance FNALU is comprised of several different platforms. UNIX exists in different *flavors* on the different vendor platforms, meaning that the UNIX operating system is somewhat customized to each platform. The OS is actually named differently on each platform: AIX on IBM, IRIX on Silicon Graphics, Solaris on Sun, Digital UNIX on Digital[1], and LINUX on PC. The differences between flavors tend to be small.

---

1. See section 1.4 for a note on the OS name for Sun and Digital systems.

Customizable Environment

The "brave new world" of UNIX allows you to set up your work environment and choose your tools to suit your own individual needs. At Fermilab we have been careful to implement and support what we consider to be the most useful of the wide range of available UNIX utilities appropriate for our environment. Still, you will find a much larger array of options than you may be accustomed to. We provide you with defaults for most options, thereby allowing you to get used to the new system gradually, and build your knowledge and proficiency at your own speed.

In order to take full advantage of the flexibility offered, you need to be aware of the options, and you need to make some choices. Two choices you as an individual user need to make initially are the editor and the mail system you want to use. Of course you can change to other ones later. After that, you have the choice of delving further into the marvels of UNIX, or accepting the provided defaults so you can just get to work. UNIX is in fact usable by ordinary mortals as well as by sophisticated hackers! Throughout this manual we give some guidance for making choices.

The available editors come in three basic types:

- modern point-and-click style editors (for graphics-capable terminals) similar to those available for PC Windows and Macintosh
- UNIX editors that provide powerful features for sophisticated editing needs, but that can also be used for simple editing tasks using a small subset of commands
- VMS-style editors

The types of mail systems offered include:

- basic UNIX mail readers
- more fully functional "traditional" mail systems
- modern point-and-click mail readers

Instead of a single command interpreter and user interface, several shells have been developed for UNIX, a number of which we provide and support. Until you get a good idea of what the advantages are of one shell over another, we recommend that you just use the default shell provided. You can change it later.

The choices that you need to make should not be overwhelming. Most people find the range of choices empowering once they're familiar with the system.

1.3 The Fermi UNIX Environment

The on-site UNIX[1] computer systems at Fermilab are used for a wide range of tasks, including general purpose interactive use, batch jobs, farms[2], and data acquisition systems, among other things. In order to most efficiently accommodate the needs of the Fermilab user community, the Computing Division provides a single, operating system-independent UNIX environment known as the *Fermi UNIX Environment*, or *FUE*.

The Fermi UNIX Environment (FUE) is a set of products installed on most UNIX systems at Fermilab that defines the computing environment. One of the main goals of FUE is to provide as much as possible the same environment on the different UNIX platforms. A second important goal is to provide a standard product support methodology.

---

1. UNIX has been customized to several different hardware platforms, and each of these adaptations (SunOS, IRIX, etc.) is considered a separate operating system (also called OS type in this document).
2. Farms are clusters of RISC-based workstations for which parallel processing software has been developed to support activities such as event reconstruction and Monte Carlo.

The methodology and infrastructure for product support and distribution under FUE is provided via a software support toolkit called UNIX Product Support (**UPS**). **UPS** was developed with the goal of providing a uniform and consistent interface for the management, distribution, installation and use of all the supported UNIX software. The **UPS** methodology is fully described in *UNIX Product Methodology at Fermilab*, document number GU0014. Chapter 10 of the present document describes the operations that installers and end users of the supported software need to know how to perform. Appendix B provides an overview of **UPS** to supplement the information in Chapter 10.

FUE consists of three parts:

- A set of login scripts, called the *Fermi files* that provide a common environment. These are described in section 9.4 and listed in Appendix C.
- Infrastructure to allow access to software products under **UPS**.
- General utilities, system administration tools, and software products in **UPS** format.

UNIX systems at the lab or at users' home institutions can be set up to be FUE-compliant. There are different levels of FUE compliance available; these levels are described in GU0014.

1.4 UNIX OS Support at Fermilab

The Fermilab Computing Division supports many different flavors of UNIX operating systems. The Computing Division currently supports the following platform and operating system combinations:

- Silicon Graphics running IRIX
- IBM running AIX
- Digital running Digital UNIX (OSF1)
- Sun Microsystems running Solaris (SunOS)
- Intel PCs running redhat LINUX (this is just becoming supported as this manual goes to print)

There is a command **`uname`** (also **`funame`** at Fermilab) that returns information about the current system. The OS name returned by these commands for Sun is SunOS and for Digital is OSF1. Note that these are different from the names under which the vendors now market these operating systems.

For the currently supported releases of the operating systems, C compilers and FORTRAN compilers, refer to the document DR0010 (see section 1.6.1 for information on how to locate Fermilab documents). This information changes frequently.

1.5 The Central Fermilab UNIX Resource: FNALU

FNALU is a UNIX computing resource which is available to the laboratory at large and managed by the Computing Division. It provides batch and interactive computing on IBM, Silicon Graphics, Sun, and DEC platforms. The primary file system on FNALU is AFS, described in Chapter 7. AFS is designed explicitly to serve as a distributed file system both within a local area network (LAN) and a wide area network (WAN). FUE is installed on FNALU, and it handles the provided software products in a way that appears the same to all users of the cluster, regardless of platform.

1.5.1 Getting an Account on FNALU

Accounts on the FNALU cluster fall into two categories, based on the user's resource requirements. If your resource requirements are small (< 50Mbytes/1% CPU), you can request an account using the *Computer Account Request Form*. To request an account allowing the use of resources beyond this limit, you need to submit a *Project Request Form* on which you indicate your estimated resource requirements. Both forms are accessible from the Computing Division home page in *Forms* under the heading **Services**. Read the *Proper Use of the Computing Facilities Form* (also in *Forms*) before filling out the appropriate form. Send your completed form to *compdiv@fnal.gov*, or deliver it to the Computing Division administration area, on the Wilson Hall 8th floor cross-over.

Refer to the *FNALU User's Guide*, document number GU0008, for a description of the machines that comprise the FNALU cluster and the available software.

1.5.2 Intended Uses of the FNALU Cluster

FNALU is intended to be a general-purpose, multi-platform cluster with a focus on physics and engineering requirements. The intended uses of FNALU can be summarized into a few categories:

- software repository/server and code development platform for work groups in a distributed environment
- platform for proprietary products available for users on a trial basis
- platform for users with low requirements who cannot purchase UNIX system
- distribution platform for Computing Division products and software products used directly by client systems participating in the AFS file system
- mail reader

1.6 Additional Documents and Information

1.6.1 Fermilab Documents

Unless stated otherwise, the Fermilab documents that we reference throughout this manual can be found on the World Wide Web (a.k.a. *WWW* or *the Web*), an on-line information resource to which we provide an introduction in section 3.2.1. A limited selection of documents is also available in printed form on the shelves outside the former Computing Division library, Wilson Hall, 8th floor, NE corner.

As a general rule throughout this manual, we do not provide the URL[1] for each referenced Fermilab document due to the volatility of the addresses, we only state the title and document number. You can find the documents in the Computing Division document database as described below, where they are maintained.

In addition, lots of information has been gathered under the *UNIX Resources* Web page, organized by topic. This page is linked under the heading **Documentation & Software** on the Computing Division home page.

The principal URLs you may need to reference while navigating the Computing Division Web pages are listed in section 1.6.2.

---

1. *URL* stands for Universal Resource Locator. A URL is a document's address on the World Wide Web; URL is defined in section 3.2.1 under **WWW Basics**.

Computing Division Document Database

The database contains both UNIX-related and non-UNIX documents. Most of the documents are available for viewing and/or printing in either HTML (see section 3.2.1) or PostScript format. A keyword search facility is provided, which currently allows a one-word entry.

Starting from the *Fermilab at Work* Web page, you can reach the Computing Division document database by selecting in turn:

1) *Computing Division* under the heading **Divisions & Sections** (or select *Computing* under **Work Resources**) to reach the Computing Division home page

2) *Documentation* under the heading **Documentation & Software**

From the document database, under the heading **Computing Division Documents**, you can find documents in the categories of General Documentation (document numbers starting with GG), General UNIX (GU), Division Recommendations (DR) and some other designations.

Software Products Information

Also from the document database, under the heading **Product and Application Information**, you can find documentation for many Fermilab software products. The document numbers correspond to the product numbers, and start with PM (Physics and Math Programs), PU (Utility Programs), and other designations.

Alternatively, you can find UNIX product documentation from a variety of sources directly by product name. Starting at the Computing Division home page, first select *UNIX Applications* under **Documentation & Software**, then select *product documentation* under the heading **Additional Resources and Documentation for UAS Products** to reach the *Fermilab Product Documentation Area*.

1.6.2 Useful URLs

Fermilab home page:

```
http://www.fnal.gov/
```

Fermilab at Work page:

```
http://www.fnal.gov/faw/
```

Computing Division home page:

```
http://www.fnal.gov/cd/
```

Computing Division document database:

```
http://cddocs.fnal.gov/cfdocs/productsDB/docs.html
```

KITS Product Status Report:

```
http://www.fnal.gov/docs/products/kits-report.html
```

UNIX Resources page:

```
http://www.fnal.gov/cd/UNIX/UnixResources.html
```

Fermilab Product Documentation Area page:

```
http://www.fnal.gov/docs/products/
```

1.6.3 Commercially Available Texts

There are several good books on UNIX. The vendors also have UNIX documentation, for example the Silicon Graphics *IRIS-4D User's Guide* and Sun's beginner's guides. Printed reference documentation for the commands is typically just hard copy of the UNIX on-line help, known as *man pages*, described in section 3.1. Some popular books are:

- *UNIX System V: A Practical Guide* by Mark Sobell, published by Benjamin/Cummings
- *UNIX in a Nutshell* by Daniel Gilly et al, published by O'Reilly & Associates, Inc.
- *UNIX for the Impatient* by Paul W. Abrahams and Bruce R. Larson, published by Addison-Wesley
- *UNIX for VMS Users* by Philip Bourne, published by Digital Press (Note that it has a predictable and detectable bias towards Digital's flavor of UNIX!)

The O'Reilly & Associates, Inc. publishers provide specialized texts on many UNIX applications in addition to their general UNIX texts. These are generally regarded as excellent references.

From the *UNIX Resources* Web page, see *Recommended books* under **The UNIX Operating System** for information on what's available in the stockroom. These books can be found in many bookstores, too.

Chapter 2: Getting Started on a UNIX System

This chapter describes the login and logout procedures, and discusses some basic UNIX features including the prompt, special keys, and special characters. We present a brief discussion of the file systems you are likely to encounter at Fermilab, and how to change your password according to your file system and installation.

2.1 Logging In

There are different ways to access a UNIX system:

- Log into the workstation console directly (in other words, it is on or under your own desk). Consult the instructions for your workstation.
- Connect to a UNIX machine over the network; enter[1]:

  `telnet` ***host***

- For X-terminals see section 9.5.

The system will prompt for your *login name* (or *username*) and for your *password.* Below is a sample login session from an X terminal to node FSGI01 of the FNALU system (note that the system prompt is set to the node name under FUE):

```
Connecting to host "fsgi01".....success.


IRIX System V.4 (fsgi01)

login: username
Password:(password is not shown)
IRIX Release 5.2 IP7 fsgi01
Last login: Tue Oct 24 09:27:14 by UNKNOWN@dcdx03.fnal.gov
Terminal Type is xterm


There are no available articles.
<fsgi01>
```

The login name must be all lower case characters, but the password may contain upper and lower case characters. Be sure to enter them in the correct case, because **UNIX is case sensitive**. Note that if you do use upper case to log in, UNIX may assume you have an upper-case-only terminal and you will have very limited capability. If you do so, either log out and log back in again, or enter the command:

---

1. **telnet** is described in section 13.2. Also described there is the utility **rlogin**, which can be used instead of **telnet**.

```
% stty -lcase
```

When you log in, a series of login scripts is run to define the functionality of your terminal and to set up your environment. These start-up files have been customized under FUE and are called the *Fermi files*. They are listed in Appendix C. The Fermi files provide the common environment described in Chapter 9.

At login, the system will type out:

```
Terminal Type is {termtype}
```

where `{termtype}` represents the best guess depending on exactly how and from where you entered the system. To change this terminal type, see section 9.7.

2.1.1 C Shell Family

The C shell runs two files when you log in, first a file named `.cshrc` and then a file named `.login`. They are both located in your home directory. See section 9.4.1 for details.

2.1.2 Bourne Shell Family

In the Bourne shell family the `.profile` and `.shrc` files, located in your home directory, are run when you log in.[1] Sometimes **ksh** runs `.kshrc` in place of `.shrc`. See section 9.4.2 for details.

2.2 Logging Out

The logout commands are slightly different for the two shell families. For either one, you can create a `.logout` script and use it in place of the standard commands.

If you are on a system running AFS, it is best to issue the command **`unlog`** before logging out. See section 7.3.4 for an explanation.

2.2.1 C Shell Family

You can logout with either:

```
% logout
```

or

```
% exit
```

If you have other processes running you will be informed that you have stopped jobs. You can continue to enter **`logout`** until all the processes are terminated.

The control character **eof**, which is usually set to <Ctrl-d>, is the normal UNIX way of logging off. Since this is easily entered accidentally, the standard Fermi UNIX Environment disables this way of logging out by including the command **`set ignoreeof`** in the Fermi files.

---

1. Bourne shell (**sh**) does not run `.shrc`. Sometimes (more common recently) vendors have linked **sh** to **ksh** (see section 9.4.2), effectively replacing **sh** with **ksh**.

2.2.2 Bourne Shell Family

You can logout with:

```
$ exit
```

or (on some systems):

```
$ <Ctrl-d>
```

The control character **eof**, which is usually set to <Ctrl-d>, is the normal UNIX way of logging off. Since this is easily entered accidentally, the standard Fermi UNIX Environment includes the command **`set ignoreeof`** in the Fermi files to disable it (*ignoreeof* is not supported on all Bourne shells).

If you have other processes running you will be informed that you have stopped jobs. You can continue to enter **`exit`** until all the processes are terminated.

2.3 The UNIX Prompt

After you log in successfully, you will get a prompt. The default UNIX prompt usually indicates your default shell (see Chapter 4 for information on shells). Typically, a **`%`** indicates the C shell (**csh**) and a **`$`** indicates the Bourne shell (**sh**). The Fermi files set the prompt to the machine name, which is most likely what you'll see.

You can change your default prompt by altering your start-up files.

In the C shell, include in your `.login` file:

```
set prompt='newprompt '
```

In the Bourne shell, you need to set the value of the keyword shell variable *PS1*. This is just a variable that is declared and initialized by the shell at start-up. Include in your `.profile` file:

```
export PS1; PS1="newprompt "
```

2.4 Special Keys

Like all systems, UNIX has a number of special keys that perform particular functions. Some important ones are the keys necessary to backspace over a character when entering a command, to delete the whole line being entered, and to interrupt execution. These are user-specifiable, and have different defaults based on the shell, the version of UNIX, and the login files described in section 9.4. If you are using the Fermi files (most UNIX systems at Fermilab have FUE installed and thus do use these files), these special keys will be set as described in the table (notice the footnote labelled *a* that follows the table):

Name	Control Char	Function
erase	DEL or back-space	Erase character. Backspace and erase one character (the key used depends on terminal setting). Sometimes, especially within **tcl/tk** applications, you must use <Ctrl-h>.
werase	<Ctrl-w>	Delete the rightmost word typed in.
kill	<Ctrl-x>	Kill (erase) the line typed in so far.[a]
intr	<Ctrl-c>	Interrupt the program currently running.
rprnt	<Ctrl-r>	Reprint the line typed in so far.
flush	<Ctrl-o>	Stops terminal output until you press a key.
susp	<Ctrl-z>	Suspend the program currently running and put it in the background. **This does not stop the process as it does in VMS!**
stop	<Ctrl-s>	Stop the display. To resume, press the **start** key
start	<Ctrl-q>	Start the display after stop.
eof	<Ctrl-d>	Send the program an end-of-file character.

a. If you prefer to use <Ctrl-u> for this function, uncomment the line `#stty kill '^u'` in your `.login` file (C shell family), or `#stty kill` in your `.profile` file (Bourne shell family).

To display the current settings for your terminal, enter:

```
% stty -a
```

If the keys don't seem to work as described here or you want to change them, refer to section 9.7.

UNIX relies on the hardware tabs of your terminal. If they are not set or if they are set in an unusual way, displays may appear strange on your terminal. You can set the tabs manually on your terminal, or you can use the **tabs** command to set them. The command with no arguments:

```
% tabs
```

will set tabs in the usual UNIX way, 8 spaces apart.

Special Note Regarding Backspace and Delete Keys

In some unusual circumstances of setup and keyboards you may also need to issue this set of commands to get the backspace key to work as expected:

```
% stty erase "^?"
% stty intr "^C"
% stty kill "^X"
```

Sometimes there is trouble with the delete key. Adding the following text to your `.profile` or `.login` will make the key useful, even on X terminals from different places on different platforms:

```
case $TERM in
vt100)
    stty erase \^? ;;
xterm)
    case "`xdpyinfo | grep 'vendor string'`" in
        *DigitalEquipmentCorp*)         stty erase \^? ;;
        *Network\ Computing\ Devices*)  stty erase \^H ;;
        *Silicon\ Graphics*)            stty erase \^H ;;
        *)                              stty erase \^H ;;
    esac
    ;;
esac
```

Special Note for ksh Users Regarding Arrow Keys

To make the up and down arrow keys work and therefore to enable command line editing and recall in **ksh**, include the following lines in your `.shrc` file (or `.kshrc`):

```
set -o emacs
alias __A='^P'
alias __D='^B'
alias __B='^N'
alias __C='^F'
```

Note that the `A`, `D`, `B`, and `C` are preceded by *two* underscores, and that you need to insert an actual control character, not simply carat-P or carat-B. A control character typically needs to be preceded by a "quoting" character, which differs from editor to editor.

For this (these) editor(s):	... enter this immediately before the control character:
vi	<Ctrl-v>
emacs	<Ctrl-q>
nu/TPU, fermitpu	<Ctrl-v> <Ctrl-v>
EDT+	<Ctrl-[>
NEdit	Use Insert Control Character from the Edit menu.

We believe this prescription works on all UNIX operating systems, regardless of how you're connected (e.g., **telnet**, **xterm**).

2.5 Special Characters (Metacharacters)

Slashes

The backslash (\) character is used on the UNIX command line to mask the special meaning of the character immediately following it (no spaces in-between) so that the command interpreter takes the character literally. It is called a *quoting* character. For example:

% *command* \<*CR*>

causes the carriage return to be ignored, allowing you to continue typing your command on the following line.

The forward slash (**/**) character is the symbol for the root directory. In path names it acts as a separator between directories in the hierarchy, and between the last directory and the file, if one is specified. For example:

/rootdir/ . . . /userdir/subdir1/subdir2/filename

Quotes and Parentheses

Different types of quotes have special meanings.

Normal single quotes (apostrophes) around a string (**'string'**) tell the command interpreter to take the string (**string**) literally. Double quotes around a string (**"string"**) also tell the command interpreter to take the string literally, but allow interpretation of variables that follow a **$** character (**$** preceding a variable name outputs the value of the variable; see section 9.1). Section 5.1.2 further explains single and double quotes in command interpretation, and provides an example.

Single backquotes around a command string (**`string`**) tell the interpreter to run the command(s) in the string, and to use the output of the command(s) in place of the string itself. This is useful for combining two commands into one, and for doing iterative tasks within shell scripts (shell scripts are introduced in section 4.4).

A string of commands enclosed in parentheses (e.g., **(*command1*;*command2*)**) is run in a subshell. In section 5.1.1 we discuss the difference between shell commands and non-shell commands. A non-shell command always runs in a separate shell (a subshell); when enclosed in parentheses, the command starts a second subshell.

Command Separators

The semicolon (**;**) character separates successive commands on a single command line. For example,

% ***command1*** **;** ***command2***

executes ***command1***, and when it finishes, ***command2*** gets executed.

The ampersand character (**&**) is similar to **;** but does not wait for ***command1*** to finish.

A double ampersand (**&&**) runs ***command2*** only if ***command1*** was successful.

Piping commands (the pipe symbol **|**) is discussed in section 5.4.3. A pipe tells ***command2*** to use the output of ***command1*** as input.

A double pipe (**||**) runs ***command2*** only if ***command1*** was *unsuccessful.*

Other Special Characters

Special characters are used in file expansion (section 6.2.2). Note that to prevent file expansion, these characters must be prefaced by a backslash (****). Metacharacters are also used in input/output redirection (section 5.4.2), and in regular expressions (section 5.4.5) as wildcards, delimiters, and other special pattern-matching characters. Refer to these sections for specific information.

2.6 File Systems: Standard UNIX and AFS

The standard UNIX file system is a hierarchy of directories descending from what is known as the *root directory.* UNIX allows parts of the directory hierarchy, also called file systems, to reside on separate storage devices or in separate disk partitions. These are accommodated by means of

mount points. A *mount point* is a directory in a file system that corresponds to the root directory of some other file system. The primary file system is the one starting at the true root. The standard UNIX file system is described in Chapter 6.

If the UNIX machine that you work on is part of an integrated system of UNIX machines, for example a LAN (local area network), it is likely that a distributed file serving system has been installed on it. A distributed file system provides a common directory structure and thus the same view of the file system to all participating nodes. This overrides the standard UNIX file system.

Fermilab is using the AFS (Andrew File System) as a distributed file service model, and it is installed on several machines at Fermilab in a production environment, including the FNALU cluster. See Chapter 7 for a discussion of AFS.

A special note to CDF and D0 users: Your UNIX systems are *not* configured to use AFS.

2.7 Information Distribution System: NIS

NIS (Network Information System) is a system that distributes information throughout a cluster. You may also know this by the name *yellow pages*.

Note that the word *cluster*, as used in this manual in reference to UNIX, is not the same as a VAX cluster. We define a UNIX cluster as a group of machines that share both a common password file (or user database), and a common file system, especially for login directories. NIS is usually used to provide the common password file, and the common file system is typically NFS or AFS.

NIS is installed on FNALU and many other UNIX clusters at Fermilab. In order to determine if NIS is running on your system, execute the command:

```
% domainname
```

If it returns a value, then NIS is running on your cluster. If no output is returned, then it is not. Many UNIX clusters use NIS to share a common login area across several machines. Note that it is possible for both AFS and NIS to be installed on a single system.

2.8 Changing Your Password

For any password changes, you will be prompted for your old and new passwords. Your password must be at least six characters long and should not be your login name or any simple permutation of it. We recommend that you limit your password to eight characters, especially if it is used as an AFS password (see section 7.3.2). It is advisable to mix letters and digits in your password.

If your system is configured in a way not covered in the following sections, you may need to contact your local system manager to determine which password(s) to change.

Man pages are available for the commands described below: **`passwd`**, **`yppasswd`**, and **`kpasswd`**.

2.8.1 Standard UNIX Password

This section assumes that AFS is not running on your system.

Your UNIX password is used to log you into the system. This password is stored in one of two ways, depending on the configuration of your system.

If your system uses the standard UNIX file system, and NIS is not installed, your UNIX password is part of your local password file. In this case you need to execute the generic UNIX password-changing command to change your password:

```
% passwd
```

The system will prompt you for the necessary information. Your password is only changed for the machine, or node, on which you execute this command.

If NIS is running on your system, your UNIX password is stored in the NIS password file which is shared by all machines on the NIS cluster. In this case, you need to use the command **yppasswd** to change your password. The system will prompt you for the necessary information. When you change your password in this way, it changes for all the machines (nodes) that are part of your cluster.

2.8.2 Kerberos (AFS) Password

Read section 7.3.2 to understand how a Kerberos (AFS) password works. You can change your Kerberos password using the command:

```
% kpasswd
```

The system will prompt you for the necessary information.

This command changes your Kerberos password for all systems that run AFS on-site.

Chapter 3: Information Resources

This chapter introduces you to the information facilities available from UNIX. Standard UNIX on-line help is available via the man pages. We discuss the World Wide Web and newsgroups, which are very rich sources of information on a virtually unlimited set of topics. A few utilities that allow you to get information about vendor products, other users, and the Fermilab computing systems are also covered. Finally, we include instructions for communicating with the Fermilab Helpdesk.

3.1 UNIX On-Line Help

3.1.1 man Pages

On-line help for UNIX system commands and utilities is in the form of *man pages* (*man* stands for manual) which consist of an on-line version of the UNIX documentation set (often called the UNIX Programmer's Reference Manual). You access the man pages with the **man** command.

Note that the man pages differ in many instances between UNIX platforms.

The man Command

When you need help on a known command, use the general **man** command format:

% man [*part*] *topic*

where ***topic*** is generally a UNIX command. **man** is really the on-line manual which is divided into several parts. ***part*** is a digit between 0 and 9. If you know in advance which part contains the information you want, you can speed the search by specifying it. More often than not you will just enter:

% man *topic*

The word *print* in **man** entries usually means *display on the screen*. Don't be confused by this. Several options are available with the **man** utility, described under **man man**. The **man** command normally displays complete manual pages that you select by name. One-line summaries can be selected by either by keyword (**-k** option), or by the name of an associated file (**-f**). These options are described in section 3.1.2.

A typical initial **man** screen can be seen by issuing the command:

% man ls

where **ls** is the UNIX command to list files in a directory.

```
ls(1)                    User Commands                    ls(1)



NAME
     ls - list contents of directory

SYNOPSIS
     ls [ -abcCdfFgilLmnopqrRstux1 ] [ names ]

AVAILABILITY
     SUNWcsu

DESCRIPTION
     For each directory argument, ls lists the  contents  of  the
     directory;  for  each file argument, ls repeats its name and
     any other  information  requested.  The  output  is  sorted
     alphabetically  by  default.  When no argument is given, the
     current directory is listed.  When  several  arguments  are
     given,  the  arguments  are  first sorted appropriately, but
/tmp/mpa002Zf
```

Man pages are typically formatted with the UNIX text processing utility **nroff** (or **groff**). These utilities are covered in most UNIX texts. If you find that the **man** page is unformatted, run **`setup groff`**, and then rerun the **`man`** command.

Note that built-in shell commands are described under the topic corresponding to their shell.[1] (See section 5.1.1 for information on built-in commands.) For example, to get information on the command **`alias`** for your current shell, you would enter **`man`** ***shell*** (e.g., **`man bash`**) and search there for information on **`alias`** using the ***/pattern*** function described below.

There is an alternate, "quick and dirty" method to verify the format of a command and get a listing of its options. It doesn't work with all commands, but is usually worth a try. Simply enter the command with an illegal option (try **`/`** or **`?`** or **`.`**). For example,

```
% ls -/
```

will produce the output:

```
ls: illegal option -- /
usage: ls -RadCLHxmnlogrtucpFbqisf [files]
```

The man Command for AFS Commands

For AFS commands, it works a little differently. AFS commands are discussed in section 7.4. The **man** page for an AFS command is found by entering:

```
% man fs_command
```

Note the underscore (_) between **`fs`** and the rest of the AFS command; the underscore is only used with the **`man`** command.

---

1. Some platforms provide **man** pages for built-in commands, however in general you may find it easier to look in a reference book!

Manipulating man Pages

man displays the information using your *$PAGER* environment variable, which under FUE is set to **less** (see section 5.4.4). Therefore, **man** pages are normally piped to **less.** The command **`man less`** will give you more information about manipulating **man** via the **less** filter. **less** gives you one page at a time and lets you enter commands at the prompt to control what it does after each page. For example, you page forward with the Spacebar and page backward using **`b`**.

Once in the **man** environment, you can search for patterns by entering the ***/pattern*** option at the command line. The first instance of the string ***pattern*** will appear in the top line of the screen. To find additional instances of the pattern in the text, simply enter a slash (**`/`**).

To exit from **man** enter **`q`** (for quit).

Printing man Pages

To print a **man** page, you can use the pipe feature (**|**) along with recommended print formatting and printing commands. These features are covered in Chapter 8. As a suggestion, pipe the output of the **`man`** command to **`a2ps -m`** (to convert **man** pages to PostScript format) and then pipe that output to the print command **`flpr`**:

`% man` ***command*** **`| a2ps -m | flpr`** **`[`*****options*****`]`**

This formats the output nicely in landscape, two pages to a sheet.

3.1.2 Finding the Right Command

If you don't know exactly what command you need, try the **`-k`** option with a keyword.

`% man -k` ***keyword***

This displays the **man** page name, the section number in the UNIX documentation, and a short description for each **man** page whose name line contains ***keyword***. For example to find a search utility, enter:

`% man -k search`

The system will output several records similar to the following:

```
conflict(8) - search for alias/password conflicts /usr/local/lib/mh/conflict [-m
glookbib(1) - search bibliographic databases glookbib [ -v ] [ -istring ] [ -tn
ident(1) - identify files ident [ -q ] [ \&.\|.\|. ] ident searches for all occu
lkbib(1) - search bibliographic databases lkbib [ -v ] [ -ifields ] [ -pfilename
lsearch(l) - See if a list contains a particular element lsearch ?mode? list pat
lsearch(n) - See if a list contains a particular element lsearch ?mode? list pat
zgrep(1) - search possibly compressed files for a regular expression zgrep [ gre
..
..
```

Some UNIX systems have an additional utility, **apropos**, which can be used to locate commands by keyword lookup:

`% apropos` ***keyword***

`apropos` ***keyword*** is equivalent to:

`% man -k` ***keyword***

The **-f** ***filename*** option for **man** prints the manual entry summaries which might pertain to the given filename(s). Any leading pathname components are stripped from the filename before the filename is matched against the summaries. Here is an example using the **-f** option, followed by the output:

```
% man -f /etc/passwd

   passwd (1)              - change login password and password attributes
   passwd (4)              - password file
```

The summaries are gotten from the **whatis** database. You can run **whatis** ***command(s)*** to look up a given command and obtain the header line from the manual section. You can then run **man** to get more information on the command. If the line starts `name(section)...` you can do **man** ***section name*** to get the documentation for it.

3.1.3 Vendor Product Documentation

Most vendor product documentation is now available on-line, viewable with an X-terminal or workstation.

Flavor:	Command:
AIX	**info**
IRIX	**insight**
Digital UNIX (OSF1)	**dxbook**
Solaris (SunOS)	**answerbook** (only usable on Sun workstations at Fermilab)

3.2 The Internet

The Internet is a global network of networks that provides access to hundreds of thousands of computers around the world. As the reach of the network has grown, so has the number of services accessible. The main tools that allow the user to navigate through the Internet are:

telnet	to access remote hosts (see section 13.2)
ftp	to retrieve data files (see section 13.1.1)
Mail	to send mail (note that Web browser mail handlers are *not* covered in Chapter 12)
WWW	to browse the World Wide Web (see section 3.2.1)
News	to scan the numerous Usenet news groups (see section 3.2.3)

There are two ways to reference an Internet host: an alphabetic name and a series of numbers. The alphabetic sequence is called the *host name* (e.g., fnsg01.fnal.gov) and the numeric one is called the *IP address* (e.g., 131.225.8.178). At Fermilab all host names end with *fnal.gov*, where this suffix is called the *domain name*. Since hosts may change their IP addresses, it is a good practice to always use the host name.

An excellent introduction to the Internet services is *The Whole INTERNET, User's Guide and Catalog*, published by O'Reilly & Associates.

3.2.1 The World Wide Web (WWW or "the Web")

Our primary way of delivering information to you, especially relatively static information, is via the World Wide Web (also known as WWW or "the Web"). There are a number of reasons why the Web has become the defacto standard for information delivery at Fermilab, within the HEP community, and even in the computer-literate parts of the world of business:

- The Web is very strong at pulling together information from different places, and of different formats and types (native HTML[1], PostScript, newsgroups, and even VAXNEWS), and making it appear as a seamless whole.
- It is relatively easy to make information available on the Web.
- Browsers (the programs you use to view information on the Web) are available for every supported platform at Fermilab, making it an excellent fit to the distributed environment in which we now work.

Accessing the Web

In order to access the Web, enter (or put in your login script):

```
% setup www
```

Web browsers work best on workstations that support graphics, so if possible you should have on your desktop a workstation, X terminal, PC or Macintosh rather than a "dumb" terminal. If you have a graphics terminal, use either **Netscape** or **Mosaic**[2]. To invoke them on an X terminal enter:

```
% netscape [&]
```

or

```
% mosaic [&]
```

The ampersand (`&`) is discussed in section 5.5.2. In order to use an X application (which these products are), you must have defined your *$DISPLAY* variable correctly. See section 9.2.

If you don't have a graphics terminal, then you must use one of the line-mode programs, **lynx** or **www**. **lynx** is generally thought to be the better of the two. To invoke them, enter:

```
% lynx
```

or

```
% www
```

This section is not intended to provide detailed instruction on the use of any particular Web browser. Once you get any of them running (at least the graphical ones) there is more information than you will ever want available under Help. In addition, from the *UNIX Resources* Web page you will find information both on using browsers and on the files that control what filetypes the browser recognizes. These files are called `.mailcap` and `mime.types`; see section 9.6 for information on them.

---

1. HTML stands for HyperText Markup Language, the standard language for documents accessed on the Web.
2. Development on **Mosaic** ended in 1996. Many newer features of the Web are not available on **Mosaic**, and in some cases will cause it to crash. In particular, **Mosaic** does not support tables. We recommend using **Netscape**.

Web Basics

Web browsers find information based on URLs (Universal Resource Locators) which are like addresses and which take you to the top level of a Web site, often called a *home page*.

A home page usually refers to the first page of a commercial, educational, government or personal Web site. Each home page can have several layers or pages that it links to, thereby creating a whole Web site. But the home page is generally the first place you would look. It is like looking at the cover of a book and its table of contents at the same time.

The native WWW addresses are of the form:

```
http://address/
```

or

```
http://address/something.html
```

The first part is the protocol, `http` in this case. (A protocol is a set of rules computers observe to exchange information. `http` stands for the HyperText Transfer Protocol; think of it as the identifier of a Web page.) Instead of `http` you might also see `ftp`, `file`, `gopher`, `mailto`, and `news`.

Next you'll see a colon and two forward slashes (except for `mailto`, which has a different format).

Next comes the computer's address in the format described in section 3.2. A computer address, or domain name, is used by computers in routing data across the many networks that make up the Net.

Finally, you often see a directory path at that computer plus a file at the end of the path. Web page files usually end in "`html`", for HyperText Markup Language, although you may also see "`htm`", "`shtm`", or "`shtml`". HTML is the simple scripting language that tells browsers how to display the various elements of a Web page such as links, body text, header text, inline graphics, and external files.

Many HTML files contain links to other documents. Sometimes links are text; sometimes they are images. If a link consists of text, it is underlined and may be in color. You can tell your cursor is on a link when a URL appears at the bottom of the screen. If a link consists of an image, you'll see a URL when you move your mouse pointer across it.

Several Web search "engines" can help you find information you are seeking, and they vary in the number of URLs they contain in their database, how deep they go into Web sites indexing information, what they index, and how frequently they "crawl" or "walk" the Web in surveying sites. There's a valuable Swiss site at URL `http://cuiwww.unige.ch/meta-index.html` that collects several WWW search tools.

Subject directories make it increasingly easy to find information about broad subjects. An excellent directory is one called Yahoo at URL `http://www.yahoo.com/`, a well-constructed directory of tens of thousands of Web pages.

Writing your Own Web Pages

If you want to start writing Web pages in HTML format, see *Creating Web Pages*, available under *Web Help* on the Computing Division home page.

A note for Web page providers on AFS systems: Set the permissions for `system:anyuser` to `rl` on directories containing files that you want to make accessible via a Web browser. See section 7.6.2 for information on AFS directory permissions.

Fermilab Home Page

Your browser is most likely configured to have either the *Fermilab Home Page* or the *Fermilab at Work* page as your default home page. If not, the *Fermilab Home Page* can be simply accessed as: `http://www.fnal.gov/`. You may want to change your default home page to *Fermilab at Work*: `http://www.fnal.gov/faw/`. To do this, set your environment variable *WWW_HOME* as follows (environment variables are discussed in section 9.1.), depending on your shell:

C shell family:

```
% setenv WWW_HOME http://www.fnal.gov/faw/
```

Bourne shell family:

```
$ WWW_HOME="http://www.fnal.gov/faw/"; \ export WWW_HOME
```

From *Fermilab at Work* you can find information on Fermilab's different divisions and experiments, activities, schedules, etc. Navigate to the Computing Division's pages to find information on UNIX.

3.2.2 UNIX Help on WWW

The man pages can sometimes be cryptic, unwieldy, or both. As an alternative, set a bookmark in your Web browser to *UNIXHelp*, accessible under the heading **The UNIX Operating System** on the *UNIX Resources* page. Here you'll find easy-to-follow instructions on the use of many UNIX features, organized into four categories: Tasks, Commands, Concepts, and Utilities.

Also from the *UNIX Resources* page you'll find *The UNIX Reference Desk* under the heading **Other UNIX Resources**. The resources in this document are divided into the following classes: General, Texinfo Pages, Applications, Programming, IBM AIX Systems, HP-UX Systems, Unix for PCs, Sun Systems, X Window System, Networking, Security, Humor.

3.2.3 Newsgroups

Usenet News (or NetNews) is a way of communicating "articles" among people world-wide. In general, information in newsgroups is volatile information, whereas information in Web pages is of longer term. We have a server here at Fermilab which receives articles from elsewhere and posts the articles originating here. Fermilab has its own newsgroups named `fnal.xxx`. CERN's groups are prefixed with `cern` and SLAC's with `slac`. General information and especially important information is posted in `fnal.announce`. NALCAL, seminar announcements, and the like are posted in `fnal.announce.seminars`. Computing Division forms are in `fnal.announce.forms`. UNIX discussion articles are posted in `fnal.comp.unix`. There are many more newsgroups, both Fermilab groups and others, that you might find of interest.

In order to make a newsreader available on UNIX, enter (or put in your login script):

```
% setup news
```

A number of readers will be made available to you. Line-mode browser commands are **`nn`**, **`rn`**, and **`trn`**. X-based browser commands are **`xrn`** and **`knews`**. **`News`** may also be read from most WWW browsers and some mail readers. The readers keep track of the newsgroups that you are interested in ("subscribed to") as well as which articles in each newsgroup you have read; all of the UNIX readers cooperatively maintain this information, so you can use different readers at different times without losing this information.

3.3 The Info Utility

Info is a facility available to the system support people to communicate with you about events regarding the Fermilab computing systems (shutdowns, for example), or other systems-related information that is newly available. To get a list of the **Info** messages, enter the command `Info`. To read an item, enter the command with the ***nametag*** of the item found on the left side of the **Info** list:

```
% Info nametag
```

Note the capital I! If the item is more than one page, press the space bar to continue. Press `q` to quit.

3.4 Other Users: WWW Directories, finger and who

WWW Directories

From the Fermilab at Work page, directories are available to point you to information about Fermilab personnel and the high energy physics community at large. These directories typically contain general information such as email addresses, phone numbers, and office locations.

finger

finger is used to find out about other users. It searches for matching account names and first or last names, if known. Depending on the vendor implementation, it may display the name of the person associated with each account, the login name, the home directory and login shell, the contents of the file `.plan` in the person's home directory, and possibly other information such as waiting mail, and time of last login. If the person is logged in, it also may display information about his or her current sessions.

Note that each vendor has a different implementation of **finger**. In addition, for security reasons many sites disable the output of **finger** over the network. It is therefore unwise to rely on the format, content, or even the availability of **finger** as a tool for finding out about users or their accounts.

The format of the `finger` command is:

```
% finger [options] [name...]
```

where ***name*** can be a part of a personal name or a username. If you specify the option `-m`, then ***name*** is matched only to account name and not the rest of the personal name.

We encourage you to create a `.plan` file. It is just a text file in which you might include information such as your office location, phone numbers, mail station, home address, schedule, emergency numbers, and so on.

finger can often be used to look up users on a remote machine by specifying the name in the standard internet form *user@host*. This form works on VMS machines with **MultiNet** running, but in this case ***name*** must be the username; otherwise not much useful information is obtained.

As an alternative, point your favorite browser to the location `http://fnal.fnal.gov/finger/`.

who

The command `who` lists certain information about the users on your system.

```
% who
```

If used with the `-q` option, only the names of the logged in users and the number of users are displayed.

The `who am i` form identifies the invoking user. The command format is:

```
% who am i
```

There are a number of options which you can read about in the man pages.

3.5 The Fermilab Helpdesk

The Fermilab Helpdesk (a.k.a. Customer Support) is available to answer questions related to the supported computer systems and software on site. Keep in mind that its first priority is to maintain central systems and networks, and to ensure that Fermilab-supported software is available and usable. Therefore a request which impacts only one individual may not receive immediate attention.

Customer Support is in service Monday through Friday, 9:00 a.m. to 5:00 p.m. You are encouraged to use email for all communications that are not urgent.

Helpdesk Email Address

During business hours:	*helpdesk@fnal.gov*
During off-hours:	*operator@fnal.gov*

Helpdesk Web Page

You can request help and/or keep track of actions taken on your requests from this page. From the Computing Division home page, select *Customer Support* under the heading **Services**.

Helpdesk Phone Number

630-840-2345

During off-hours, you can leave a phone message, or "escape" to Data Center Services (Operations) for requests requiring immediate attention.

Helpdesk Location

FCC 1 West

Chapter 4: Shells

This chapter discusses the concept of a UNIX shell, and how to manipulate shells. It includes information on the available and recommended shells and their features. The concept of a shell as an interpretive programming language is introduced.

4.1 Introduction to Shells

The kernel is the real operating system and is loaded into memory at boot time. Typically the user never interacts directly with the kernel. The utilities are programs stored on disk, and loaded into memory by the kernel when invoked.

A shell is a utility. It is run in user mode, and does not have system privileges. You have a default shell, and you can invoke other shells. Invoking shells is discussed in section 4.1.2.

The shell is the interface between the operating system and the user. It interprets the commands you type and the keys you press in order to direct the operating system to take an action. Shell scripts, which are analogous to DCL command files, allow you to use the shell as an interpretive programming language. They are introduced in section 4.4, but a comprehensive treatment of scripts is beyond the scope of this manual.

There are two families of shells: one based on the Bourne shell (this family also includes the Korn (**ksh**) and Bourne-again (**bash**) shells), and the other based on the Berkeley/C shell. The shells themselves will be discussed and compared in section 4.2.

4.1.1 Determining Your Current Shell

There are several commands available in all the shells that furnish this information. We present four examples below with sample output for **csh**. The first three, **`echo`**, **`env`** and **`finger`**, will show only your login shell. If you have invoked another shell, these commands will not reflect the new shell. **`ps`** lists information about all your active processes.

`% echo $SHELL`

displays the value of the variable name that follows the **`$`**; example output: `/bin/csh`

`% env or printenv`

shows all defined environment variables, including SHELL; example output: `SHELL=/bin/csh`

`% finger` ***your_username***

shows user information and login shell; example output:

```
Login name: username                    In real life: {your name}
Directory: /afs/fnal.gov/files/home/room3/{username} Shell: /bin/csh
```

Finally,

```
% ps
```

shows processes, including shell; example output:

```
PID TTY      TIME COMD
6264 pts/11   0:03 csh
```

Note that on some of the more recent OS releases (e.g., AIX+4 and IRIX+6.4, and likely others in the near future) `/bin/sh` is a link (links are described in section 6.3.5) to the korn shell (**ksh**). **ksh** is a superset of **sh**, so this shouldn't present any problems for you. One difference is that your `.shrc` file (see section 9.4.2) gets sourced when you run `/bin/sh` scripts.

4.1.2 Starting a Shell

A shell is started by a login process. A new shell is also started for each invocation of a terminal window or shell script (see section 4.4). Which shell gets invoked is determined by the last field in your entry in the password entry file. In a standard UNIX file system you can display your password entry by the command:

```
% grep ^username /etc/passwd
```

(**grep** is described in section 6.4.2.; the use of **^** is explained in section 5.4.5.) To display your password entry in an NIS environment, use the command:

```
% ypmatch username passwd
```

(The NIS command **ypmatch** is not described in this manual.) Sample **ypmatch** output from the FNALU system, for which the default shell has been set to **csh**, looks like:

```
aheavey:!:6302:1525:Name of User:/afs/fnal.gov/files/home/room3/aheavey:/bin/csh
```

When you log in, the login process invokes a shell program (e.g., `/usr/local/bin/tcsh` or `/usr/local/bin/bash`) and transfers control to it. The shell displays a prompt indicating it is ready for your input. The default UNIX prompts are symbols that indicate which shell is invoked (recall from section 2.3 that your prompt is likely to be set differently):

- **%** for the C shell family
- **$** for the Bourne or Korn shells

On FNALU the prompts are set to indicate the host machine, for example **<fsui01>**, or **<fsgi02>**. At any point in your session you can invoke another copy of the same shell or a different shell by typing the shell name at the prompt, for example:

```
% csh
```

invokes **csh** (C shell). This new shell, or "subshell", sits on top of your current shell. The execution of the original shell is then suspended (the shell is put to sleep), and the new shell takes control. Upon quitting the new one, the original shell wakes up and resumes control.

The average user at Fermilab does not have the privilege to change the password entry file. Therefore, to change your default shell you will need to ask your system manager.

4.1.3 Exiting a Shell

To exit a shell and return to the calling shell, type **exit** at the prompt. Repeat the **exit** command once for each subshell; when you reach your initial shell, your terminal emulation is closed, and the terminal window disappears. Instead of **exit** you may need to enter <Ctrl-d>.

4.2 Features of Available Shells

This section is excerpted from *Shell Choice, A shell comparison* (dated September 28, 1994) by Arnaud Taddei of CERN. His eleven-page document contains a brief description of the six major shells and provides an excellent comparison of features between the shells. It is available on the Web at `http://consult.cern.ch/writeup/shellchoice`.

Of the six major shells, four are in the Bourne family: **sh**, **ksh**, **bash**, and **zsh**; and two are in the Berkeley/C family: **csh**, **tsch**.

The most up-to-date shells are **tcsh** (Berkeley/C), and **bash** and **zsh** (Bourne). These are also the three shells that are public domain (as opposed to vendor-supported). The public domain shells are the same on all platforms, which is not true of vendor shells. This is desirable when attempting to homogenize user environments. **Note that zsh is not supported at Fermilab.**

Some of the common features of these newer shells are:

- specific startup files
- startup files are the same for any platform
- specific shell variables
- specific built-in commands

The **tcsh** is essentially an enhanced **csh**. Some additional features of the **tcsh** are:

- enhanced completion[1] mechanism (programmable for commands, file names, variable names, user names, etc.)
- multiline editing capabilities (command line editing using **emacs** or **vi**-style key bindings)
- enhanced file expression syntax
- spelling correction (see section 5.3)
- enhanced prompt
- step up/down through history list

The following table should give you an idea of the virtues of each of the shells supported at Fermilab. It is adapted from one in Taddei's document referenced above. More complete feature lists for all the shells can be found there.

++	good
+	existing
-	weak
--	absent

Criteria	sh	ksh	bash	csh	tcsh
Configurability	-	+	++	+	++
Execution of commands	+	+	+	+	++
Completion	--	+	++	+	++
Line editing	-	+	++	-	++

1. This feature allows you to uniquely specify a file without typing in its whole name.

Criteria	sh	ksh	bash	csh	tcsh
Name substitution	+	+	++	+	++
History	--	+	++	+	++
Redirections and pipes	+	+	+	+	+
Spelling correction	--	--	--	--	+
Prompt settings	+	+	+	+	++
Job control	--	+	+	+	+
Execution control	+	+	+	+	+
Signal handling	+	+	+	-	-

4.3 Supported/Recommended Shells at Fermilab

On many systems at Fermilab, **tcsh** is used as the default shell. The Computing Division currently supports **csh**, **tcsh**, **sh**, **bash** and **ksh**. **tcsh** or **bash** is recommended for interactive use, and **sh** for scripts. (The C shell family is not recommended for scripts due to inconsistent syntax at different levels of nesting.) **zsh** is not currently supported. From the *UNIX Resources* Web page you can access UNIX support information under **UNIX Products** to find out about any support policy changes.

4.4 Shell Scripts

As mentioned above, a UNIX shell can be used as an interpretive programming language. Besides executing shell commands within the script, you can:

- create and use variables
- process (read) arguments
- test, branch, and loop
- perform I/O

A *shell script* is a file containing a sequence of commands which can be executed by the shell, and flow control commands. The same syntax is used for commands within scripts as for interactive command entry. Section 5.1 explains briefly how the system runs and interprets shell scripts.

Although you can write complex programs using the shell language, you can also create simple shell scripts for running long commands or a series of commands that you use frequently.

In every shell script you write, include the special characters `#!` followed by the pathname of the shell as the first characters in the file. This indicates (a) that this is a script rather than a compiled executable, and (b) which shell to invoke to run the script.[1]

1. On some of the more recent OS releases (e.g., AIX+4 and IRIX+6.4, and likely others in the near future) `/bin/sh` is a link to the korn shell (**ksh**). Therefore on these platforms, the `.shrc` file (see section 9.4.2) gets sourced for any script starting with `#!/bin/sh`.

For example:

```
#!/usr/local/bin/bash
```

at the start of the script invokes **bash** to run it. A `#` found anywhere else in the script is interpreted as the beginning of a comment, and the shell ignores all characters between the `#` symbol and the next newline character.

An introductory reference for script-writing with examples can be found in *UNIXHelp* on the Web. You can get to it via the *UNIX Resources* page under **The UNIX Operating System**.

Note that in order to execute the script, regardless of shell, the script file must have execute permission for the appropriate users (see section 6.6.1 for a discussion of permissions). After you set this permission, the shell will need to rebuild its "hash table" to include the new script. The hash table is a table of executables that the shell recognizes.

To complete these two operations, enter:

```
% chmod a+x filename
% rehash[1]
```

To run a script, the shell must be able to locate it. If its directory is in your path (see section 9.2), you only need to type the script's filename to run it. If not, you can type the the filename preceded by `./` on the command line (the `./` explicitly tells the shell to look for the executable file in the current working directory). Typing the full path of the filename will work too, although it is perhaps the most cumbersome way of telling the shell where the script is. Here we illustrate the three ways to invoke a script:

```
% filename
% ./filename
% /full_path/.../filename
```

Once the shell locates the script, it interprets and executes the commands in the file one by one.

You may want to maintain a `$HOME/bin` directory for all your programs and shell scripts, and include this directory in your path[2]. The shell uses this variable to locate commands and other executables.

It is important to remember that, like all UNIX commands that are not part of the shell (see section 5.1.1 for an explanation of shell commands), the script file executes in a subshell forked[3] by the parent shell. This subshell retains any environment variables defined in the script as well as any shell variables defined in the file `.cshrc` or `.shrc` (one of these two files may be executed automatically prior to the script, depending on your shell; see section 9.4). At the end of the script, control returns to the parent shell, and any definitions made by the subprocess are *not* passed back to the parent process.

To execute a script for which you do want to pass back changes to the parent shell (for example, setting new shell variables), the syntax for execution differs. For the C shell family, execute the script by typing:

```
% source filename
```

---

1. The command in **sh** is `hash`; not necessary in other shells.
2. Under FUE, the Fermi files add your `/bin` directory to your PATH.
3. Under UNIX, the term *fork* means create a new process.

For the Bourne shell family, type:

$. *filename*

The **source** or **.** command executes the script in the context of your current process, so that you can affect this current process, in contrast to normal command execution.

For instance, after you make changes to your `.cshrc` or `.login` file, you can use **source** or **.** to execute it from within the login shell in order to put the changes into effect.

4.5 Other Interpretive Programming Languages

We have mentioned that each UNIX shell can be used as the interpreter for its own programming language. Other interpretive languages supported at Fermilab are **perl** (provided in the FUE **shells** product), and **gawk** (a version of **awk**; see section 5.4.4). These languages are beyond the scope of this manual. The O'Reilly & Associates, Inc. publishers provide excellent reference texts on them.

Chapter 5: Important UNIX Concepts

This chapter introduces you to the UNIX command structure, and to many important commands and concepts. The features introduced in this chapter constitute the core of the UNIX operating system, and many of these tools are quite powerful and flexible. Some of the features are shell-specific, and we provide the distinctions where necessary.

5.1 Processing Environment

5.1.1 Programs, Commands and Processes

A *program* is an executable file. A program is invoked by entering its filename (which is the *command* associated with the executable), often followed by options, arguments, and/or parameters on the command line. The shell allows three types of commands:

- an executable file that contains object code produced by a compilation of source code
- an executable file that contains a sequence of shell command lines (a *shell script*)
- an internal shell command (*built-in* command)

The first two command types may include standard UNIX utilities, commercial products, and user-written programs. All the shells allow both *interactive* command entry in which the commands are typed at the keyboard and executed one by one, and *scripted* entry in which commands are put in a file, called a shell script, and executed sequentially when the script is run. See section 4.4 for a brief discussion of the uses of shell scripts and how to execute them.

Shells execute commands by means of *processes*. A process is an instance of a program in execution. A process can interact with the kernel by invoking a well defined set of *system calls*. The system calls instruct the kernel to perform particular operations for the calling program and they can exchange data between the kernel and the process. For example, a process can use system calls to create new processes and terminate running processes.

When a terminal session begins, the operating system starts a single *parent process*. Creating a new process from an existing process is called *forking*. This new process is called a *child process* or *subprocess*. Each process has a unique process identification number (PID). A subprocess can fork another process and become a parent. A process which is not receiving input from the terminal, either running or stopped, is said to be in the *background* (see section 5.5). The **ps** command can be used to print the status of active processes. See the man pages for information about its options.

When you give the shell a command associated with a compiled executable or shell script, the shell creates, or *forks*, a new process called a *subshell*. The new process runs the system call *exec* which invokes yet another program to execute the command in place of the current process (the subshell). Unless the subprocess runs in the background, the parent process remains dormant until its subprocess completes or is stopped. Then control returns to the parent.

To execute most built-in commands, the shell forks a subshell which executes the command directly (no *exec* system call). For the built-in commands **cd**, **set**, **alias** and **source**, the current shell executes the command; no subshell is forked. You can, however, cause the shell to fork a process by enclosing the command in parentheses. The following example illustrates this (use of the semicolon is described in section 2.5; and the commands **cd** (change directory)and **pwd** (print working directory) are described in section 6.5):

`% cd /dir1; pwd`	displays `/dir1` (no subshell is forked)
`% (cd /dir2; pwd)`	due to the parentheses, a subshell is forked, then the commands are issued; displays `/dir2`. Control then returns to the parent process.
`% pwd`	displays `/dir1` since the current process was unaffected by the previous command line.

Most built-in commands exist in all shells, but there may be differences regarding arguments, options, or output format between the shell-specific versions of each command. Some commands for a given shell are not available on all platforms. Refer to a UNIX text for lists of built-in commands.

You do not need to distinguish between built-in and other commands to execute them. However in order to find help in the man pages, you do need to know which is which. Help on shell commands is usually found under the shell name, for example under **man tcsh** or **man bash**. Some platforms provide man pages for built-in commands, however in general you may find it easier to look in a reference book! Help on other commands is found directly under **man *command***.

5.1.2 Command Interpretation by the Shell

When the shell receives a command, it interprets it in a series of three (for Bourne shell family) or four (for C shell family) passes. Naturally, if the command is an alias (see section 9.3), it requires an additional pass up front for substitution.

- The first pass for the C shell family looks for the **!** character, and replaces it with the previous command (see section 5.3 for information on command recall).
- The next pass (the first pass for the Bourne shell family) replaces wildcards (used in filename expansion, redirection, and regular expressions; see sections 6.2.2, 5.4.2, and 5.4.5, respectively).
- The next pass looks for the **$** character in order to replace variable names with their values (see section 9.1).
- The final pass splits the command line elements by whitespace to arrive at the final, literal command that the shell must execute.

There are ways to prevent interpretation of special characters in each of these passes. Preceding a character with a backslash (\) works for all special characters; wildcards can be enclosed in single or double quotes; variables can be enclosed in single quotes; and whitespace is ignored if the argument containing the whitespace is enclosed in single or double quotes.

To illustrate the operations that take place in each pass, the following table presents a series of three examples using the **echo** command and the same string, first in single quotes, then double quotes, and finally with no quotes. The **echo** command writes the string to standard output. Assume that the files that match **q*** are `qq` and `qqq`, and the value of the variable *a* is **foo**.

Command -->	`echo 'q* $a x'`	`echo "q* $a x"`	`echo q* $a x`
After first pass[a], only wildcards are interpreted.	`echo 'q* $a x'` (no wildcard expansion due to quotes)	`echo "q* $a x"` (no wildcard expansion due to quotes)	`echo qq qqq $a x` (unquoted wildcard is expanded)
After second pass, unquoted or double quoted variables are replaced by their values.	`echo 'q* $a x'` (no variable replacement due to quotes)	`echo "q* foo x"` (double-quoted variable `$a` replaced by value)	`echo qq qqq foo x` (unquoted variable `$a` replaced by value)
After final pass, command string is broken up according to whitespace. The separate elements are listed vertically.	`echo` `'q* $a x'` (string treated as one argument due to quotes)	`echo` `q* foo x` (string treated as one aregument due to double quotes)	`echo` `qq` `qqq` `foo` `x` (no quotes; each argument treated separately)
When you type in the original command, the system returns the string:	`q* $a x`	`q* foo x`	`qq qqq foo x`

a. This would be the *second* pass for C shell family; there were no `!` characters to replace.

5.2 Command Entry

A UNIX command is either a built-in command or the name of an executable file which the operating system will load and execute. When you see the prompt, you can enter a command by typing the command name, any options and arguments, followed by a carriage return.

Recall, the formats displayed in this manual use **`this font style`** to indicate characters to be typed as is, and ***this font style*** to indicate arguments to be substituted. Arguments enclosed in square brackets, `[...]`, are optional.

You should be aware that UNIX commands are not noted for their consistency of format. Furthermore, commands, formats, arguments, and options may vary slightly from one UNIX flavor to another. In this manual, we attempt to be as generic as possible, and describe options that are widely available.

UNIX commands are described on-line in the man pages (see section 3.1).

5.2.1 Command Format

The basic format of UNIX commands is:

`%` ***command -option(s) argument(s)***

where:

`%`	is the (default, non-FUE) **csh** prompt.[1]
command	is the UNIX command name of a utility or tool.
option(s)	modifies how the command runs; options are nearly always preceded by a dash and listed one after another. See example below.
argument(s)	specifies data or entities (usually files) on which the command is to operate; arguments are separated by blanks ("white space").

Remember, UNIX is case-sensitive. Therefore UNIX commands must be entered in the correct case. Most of the time commands are entered in lower case.

The components are separated by at least one blank space. If an argument contains a blank, enclose the argument in double quote marks. Normally, options can be grouped; e.g., the `-lw` and the `-l -w` option specifications are equivalent in the examples below (`wc` is a sample command; it lists line, word, and/or character count of one or more files.):

`% wc -lw` ***file1 file2***

`% wc -l -w` ***file1 file2***

Some options can have arguments, and there isn't consistency on whether there should be a blank space between the option and its argument. Check the man pages when you're not sure. In the next example which shows the FORTRAN command, ***outputfile*** is the argument of the option `-o`:

`% f77 -o` ***outputfile program*****`.f`**

Looping and conditional commands are also supported. These are more advanced shell commands and are not covered in this manual. Consult a UNIX text for information on these.

5.2.2 Miscellaneous Command Line Features

- To correct typos you can use the erase key (Delete or Backspace) to erase character-by-character, or the Kill key to kill an entire line (see section 2.4).
- More than one command can be entered on a line if the commands are separated by semicolons. The commands will be executed sequentially. See section 2.5 for more information on using multiple commands on one line.
- If you need to continue a command to a new line, you can either keep on typing (without doing a carriage return), or enter a backslash (\) followed directly by a carriage return (no space in-between) and then continue typing on the next line. (Recall the backslash is used to prevent a special character's meaning to be interpreted by the shell. See section 2.5.)
- You can use parentheses to group commands. Since a subshell is created for each group, this can be used to prevent changing the current environment. It can also be used to redirect all output from the commands considered as a group (see section 5.4.2).
- Type ahead works, even if the characters get interspersed with output.

1. `$` is the non-FUE default for Bourne shell.

5.3 Command Recall

Command recall is quite different in each shell. One common feature for all shells that support command recall is the **history** mechanism. It maintains a list of commands that have been entered and allows them to be reexecuted. The *history* variable, set to some number at login time in the start-up files, determines the number of commands that are saved in the list. The *savehist* variable specifies how many commands are to be saved for your next session after you log out. The `history` command displays the list of saved commands:

```
% history
```

We discuss the following shells separately: **csh**, **tcsh**, and **bash/ksh**. There is no command recall facility for **sh**.

csh

There is no command line editing native to **csh**. Before describing the standard **csh** command recall facility, we should mention a Fermilab product called **cedit** that we recommend for use with **csh** instead. It was designed to mimic VMS line editing, and turns out to provide similar command recall and editing functionality to **tcsh**. To use **cedit**, you need to set it up initially. Enter:

```
% setup cedit
```

To execute it, type:

```
% m
```

followed by <Return>. `m` stands for *modify*. Use the up or down arrow keys to scroll to the desired command. The right and left arrow keys and your backspace key allow you to edit the command before reexecuting it. There are several control characters that perform functions within **cedit**. Typing <Ctrl-i> in **cedit** displays the available commands.

Recalling history commands using standard **csh** syntax is fairly easy. Use the commands listed below.

`!!`	Reexecute the previous command
`!`***n***	Reexecute command ***n*** from the history list
`!`***text***	Reexecute the most recent command beginning with ***text***
`!?`***text***`?`	Reexecute the most recent command containing ***text***

For example, to reexecute the 4th command from the history list, enter:

```
% !4
```

and to reexecute the last command starting with `ls`:

```
% !ls
```

The dollar sign (`$`) can be used to recall the last word of a command. `!$` causes substitution of the last word of the last command. For example, you can check the contents of `myfile.f` and then compile it using the following command sequence:

```
% less myfile.f

% f77 !$
```

A couple of nice features you can use with these reexecution commands are *preview* (**p**) and *substitute* (**s**). To substitute a string in the previous command and preview it before execution, use the syntax:

% !:p:s/*oldstring*/*newstring*

To do the same for the ***n***th command in the history list, use:

% !*n*:p:s/*oldstring*/*newstring*

To execute after previewing (and/or substituting), simply type:

```
% !!
```

tcsh

Recalling commands is easy if you are using **tcsh**. The up/down arrows on the keyboard can be used to recall commands and the left/right arrows can be used to move around within the command to edit it (VMS users will be familiar with this concept).

A command line correction algorithm is available in **tcsh**. To enable it, enter:

```
% set correct=all
```

This causes all words on the command line to be checked. If any part gets corrected, the system notifies you, and gives you a chance to accept or reject it. For example, say you type in:

```
% lz /usr/bin
```

The system will return with:

```
CORRECT ls /usr/bin (y|n|e|a|)?
```

Where `y`=yes, `n`=no, `e`=edit, and `a`=abort. You must provide one of these responses.

To turn off command line correction, enter:

```
% set correct=none
```

ksh

Two styles of command recall are supported; **emacs** and **vi**. The style is determined in one of two ways:

- include the line **`set -o`** ***editor*** in either your `.profile` or `.shrc` file, where ***editor*** is either **emacs** or **vi** (this takes precedence if variables below are set differently)
- set either the *EDITOR* or *VISUAL* environment variable to one of these editors

When set to **emacs**, use the usual **emacs** commands to display and modify previous commands, for instance <Ctrl-p> for previous line. When set to **vi**, command recall is initiated by typing the Escape (or <Ctrl-[>) key. Then all the standard **vi** commands can be used. Some of the basic **vi** and **emacs** commands are listed in section 11.3.

bash

Both **csh** and **ksh**-style recall are supported.

5.4 Important Concepts

This section attempts to provide an overview of a few of the important concepts in UNIX which are very different from other systems and may therefore be confusing to the novice user. In order to be able to make effective use of UNIX, these concepts need to be understood.

5.4.1 Path

When you issue a command, the shell program parses the command line and either processes it directly or searches for an executable file with that name in any of the directories specified in your *search path*, which is controlled by the variable *PATH*. If the file is not found in any of the directories in your search path, the shell reports that the command was not found. The file may well be on the disk somewhere, but it is **not in your path**.[1]

FUE attempts to provide an appropriate path, and we recommend that you not change this basic path. However, feel free to *add* directories to it. For the **csh** family, your `.login` file contains a `set path` line for the shell variable *path*.[2] Uncomment this line (remove the `#`) and include additional directories in the shown format:

```
set path=($path /dir1 /dir2... )
```

Or change the environment variable *PATH* (also in `.login`), as follows:

```
setenv PATH "${PATH}:/dir1:/dir2"
```

For the **sh** family, uncomment and add directories to the `PATH` line in your `.profile` file:

```
PATH=$PATH:/dir1:/dir2...
```

See section 9.2 for information on the *PATH* variable.

As an aside, if you add an executable to one of the directories in your search path, it may be necessary for you to either log out and log back in, or to recreate the internal tables used by the shell with the **`rehash`** (**csh**) or **`hash`** (**sh**) command (see section 4.4).

5.4.2 Standard Input and Output Redirection

The shell and many UNIX commands take their input from *standard input* (`stdin`), write output to *standard output* (`stdout`), and write error output to *standard error* (`stderr`). By default, standard input is connected to the terminal keyboard and standard output and error to the terminal screen.[3]

The way of indicating an end-of-file on the default standard input, a terminal, is usually <Ctrl-d>.

Redirection of I/O, for example to a file, is accomplished by specifying the destination on the command line using a *redirection metacharacter* followed by the desired destination.

---

1. This concept will be familiar to users of MS-DOS.
2. Shell versus environment variables are discussed in section 9.1.
3. VMS users would know these as the logical devices SYS$INPUT, SYS$OUTPUT, and SYS$ERROR.

C Shell Family

Some of the forms of redirection for the C shell family are:

Character	Action
`>`	Redirect standard output
`>&`	Redirect standard output and standard error
`<`	Redirect standard input
`>!`	Redirect standard output; overwrite file if it exists
`>&!`	Redirect standard output and standard error; overwrite file if it exists
`|`	Redirect standard output to another command (pipe)
`>>`	Append standard output
`>>&`	Append standard output and standard error

The form of a command with standard input and output redirection is:

% *command* -[*options*] [*arguments*] < *input file* > *output file*

If you are using **csh** and do not have the *noclobber* variable set (see section 9.2), using **>** and **>&** to redirect output will overwrite any existing file of that name. Setting *noclobber* prevents this. Using **>!** and **>&!** always forces the file to be overwritten. Use **>>** and **>>&** to append output to existing files.

Redirection may fail under some circumstances: 1) if you have the variable *noclobber* set and you attempt to redirect output to an existing file without forcing an overwrite, 2) if you redirect output to a file you don't have write access to, and 3) if you redirect output to a directory.

Examples:

% who > names	Redirect standard output to a file named `names`
% (pwd; ls -l) > out	Redirect output of both commands to a file named `out`
% pwd; ls -l > out	Redirect output of **ls** command only to a file named `out`

Input redirection can be useful, for example, if you have written a FORTRAN program which expects input from the terminal but you want it to read from a file. In the following example, `myprog`, which was written to read standard input and write standard output, is redirected to read `myin` and write `myout`:

```
% myprog < myin > myout
```

You can suppress redirected output and/or errors by sending it to the *null device*, **/dev/null**. The example shows redirection of both output and errors:

```
% who >& /dev/null
```

To redirect standard error and output to different files, you can use grouping:

```
% (cat myfile > myout) >& myerror
```

Bourne Shell Family

The Bourne shell uses a different format for redirection which includes numbers. The numbers refer to the file descriptor numbers (0 standard input, 1 standard output, 2 standard error). For example, `2>` redirects file descriptor 2, or standard error. ***&n*** is the syntax for redirecting to a specific open file. For example `2>&1` redirects 2 (standard error) to 1 (standard output); if 1 has been redirected to a file, 2 goes there too. Other file descriptor numbers are assigned sequentially to other open files, or can be explicitly referenced in the shell scripts. Some of the forms of redirection for the Bourne shell family are:

Character	Action
`>`	Redirect standard output
`2>`	Redirect standard error
`2>&1`	Redirect standard error to standard output
`<`	Redirect standard input
`|`	Pipe standard output to another command
`>>`	Append to standard output
`2>&1|`	Pipe standard output and standard error to another command

Note that `<` and `>` assume standard input and output, respectively, as the default, so the numbers 0 and 1 can be left off. The form of a command with standard input and output redirection is:

`$` ***command*** **`-[`*****options*****`] [`*****arguments*****`] <`** ***input file*** **`>`** ***output file***

Redirection may fail under some circumstances: 1) if you have the variable *noclobber* set and you attempt to redirect output to an existing file without forcing an overwrite, 2) if you redirect output to a file you don't have write access to, and 3) if you redirect output to a directory.

Examples:

`$ who > names`	Direct standard output to a file named `names`
`$ (pwd; ls -l) > out`	Direct output of both commands to a file named `out`
`$ pwd; ls -l > out`	Direct output of **`ls`** command only to a file named `out`

Input redirection can be useful if you have written a FORTRAN program which expects input from the terminal and you want to provide it from a file. In the following example, `myprog`, which was written to read standard input and write standard output, is redirected to read `myin` and write `myout`.

```
$ myprog < myin > myout
```

You can suppress redirected output and/or error by sending it to the *null device*, **`/dev/null`**. The example shows redirection of standard error only:

```
$ who 2> /dev/null
```

To redirect standard error and output to different files (note that grouping is not necessary in Bourne shell):

```
$ cat myfile > myout 2> myerror
```

5.4.3 Pipes

UNIX uses the concept of a *pipe* to connect the standard output of one program directly into the standard input of another program. This is specified by separating the two commands with the pipe operator, the vertical bar (|). The general format is:

% *command1* | *command2* | ...

where, of course, each command can have options and arguments. To implement pipes of commands, the shell forks off multiple processes. For example if you run the command:

```
% history | more
```

the shell forks twice; the grandchild runs **history**, the child runs **more** (after hooking up the right file descriptors to the right pipe ends), and the parent shell waits for the process to finish. The **history** command, a built-in, is implemented in the grandchild shell process directly, while the **more** command requires an *exec* system call.

The **tee** command can be used to send output to a file as well as to another command.

```
% who | tee whoout | sort
```

This creates a file named `whoout` which contains the original **who** output. It also sorts the **who** output and sends it to standard output, the terminal screen. The following example sends the (unsorted) **who** output to the file and the screen:

```
% who | tee whoout
```

5.4.4 Filters

A *filter* is a command or program which gets its input from standard input, sends its output to standard output, and may be used anywhere in a pipeline. Examples of filters are the UNIX utilities:

- **more** (and **less**)
- **grep**
- **awk**
- **sort**

The combination of UNIX filters **grep**, **awk**, and **sort** and the use of pipes is very powerful.

more and less

The **more** filter allows you to display output on a terminal one screen at a time. You press Spacebar to move to the following screen, and **q** to quit.

less is a much more flexible variant of the standard UNIX utility **more** and is provided under FUE[1]. The command **less** lists the output (e.g., specified files) on the terminal screen by screen like the command **more**, but in addition allows backward movement in the file (press **b** to go back one full screen) as well as forward movement. You can also move a set number of lines instead of a whole page. To view a file with the **less** filter, enter:

% less [*options*] [*filename*]...

1. FUE sets your environment variable *PAGER* to the **less** filter.

The options and usage are described in the man pages for **more** and **less.**

After displaying a page of information, **more** and **less** display a colon prompt (**:**) at the bottom of the screen and wait for instructions.

```
LESS(1)                    UNIX System V                    LESS(1)



NAME
     less - opposite of more

SYNOPSIS
     less [-[+]aABcCdeEimMnqQuUsw] [-bN] [-hN] [-xN] [-[z]N]
          [-P[mM=]string] [-[lL]logfile] [+cmd]
          [-ttag] [filename]...

DESCRIPTION
     Less is a program similar to more (1), but which allows
     backwards movement in the file as well as forward movement.
     Also, less does not have to read the entire input file
     before starting, so with large input files it starts up
     faster than text editors like vi (1).  Less uses termcap (or
     terminfo on some systems), so it can run on a variety of
     terminals.  There is even limited support for hardcopy
:
```

You can search for patterns in the file by entering ***/pattern*** at the **less** prompt. Continue to search for the same pattern using a slash (**/**). A further advantage is that **less** does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like **vi**.

grep

The **grep** filter searches the contents of one or more files for a pattern and displays only those lines matching that pattern. **grep** is described in Section 6.4.2.

awk

awk is much more than a filter; it is a powerful pattern scanning and processing language. Although you will need to spend a little time learning how to use **awk**, it is very well suited to data-manipulation tasks. It handles internally what you would have to handle laboriously in a language like C or FORTRAN. You can do in a few lines what would take many, many lines of FORTRAN.

awk works best when the data it operates on has some structure, for example a document with heading levels, or a table. In the case of a table, you can tell it the field separator (spaces, colons, commas, tabs) and it can align and interpret the contents of the field according to the way you use it. Or you can reorder the columns, or change rows into columns and vice-versa.

We present here some very basic information to get you acquainted with the concepts of **awk**, but you will need a more in-depth reference in order to use this utility. A widely-available book on **awk** is *The awk Programming Language* by Aho, Kernighan, and Weinberger, Addison-Wesley. Another good reference, from which much of the information in the present section is extracted, is *sed & awk* published by O'Reilly & Associates.

There are several versions of **awk**, and they differ from platform to platform. "Old" **awk** may be **awk** or **oawk**, "new" **awk** may be **nawk**. FUE provides a GNU version of **awk** called **gawk** as part of the **shells** product.

Some of the features of **awk** are:

- Ability to view a text file as made up of records and fields in a textual database
- Use of variables to manipulate the database
- Use of arithmetic and string operators
- Use of common programming constructs such as loops and conditionals
- Ability to generate formatted reports

With **nawk**, additional features make it easier to write larger scripts. Using **nawk** you can:

- Define functions
- Execute UNIX commands from a script
- Process the result of UNIX commands
- Process command-line arguments more gracefully
- Work more easily with multiple input streams

awk executes a set of instructions for each line of input. You can specify instructions on the command line or create a script file. Input is read a line at a time from one or more files or from standard input. The ***instructions*** must be enclosed in single quotes to protect them from the shell. (Instructions always contain curly braces which are interpreted as special characters by the shell.) We refer you to one of the books on **awk** for the available instructions. We'll use the instruction `print` in our examples.

For command lines, the syntax is:

**`% awk '`*instructions*`' `*files*

As an example, say that file `test` contains only the line `Hello, world`. The command:

```
% awk '{ print }' test
```

produces the output:

```
Hello, world
```

Multiple command lines can be entered by separating commands with semicolons or using the multi-line input capability of the Bourne shell. **awk** programs are usually placed in a file where they can be tested and modified. The syntax for invoking **awk** with a script file is:

**`% awk -f '`*script*`' `*files*

where `-f` indicates that the filename of a script follows.

awk interprets each line of the input data file(s) as a record, and each word on that line, delimited by blank spaces or tabs, as a field. You can reference these fields, either in patterns or procedures. `$0` represents the entire input line; `$1`, `$2`, ... refer to the position of individual fields on the input line. As an example, say that the file `personnel` contains a list of employees' first names, last names, and addresses. The command:

```
% awk '{ print $1}' personnel
```

would produce output of the type:

```
John
Alice
Mary
Eric
```

To use the pattern-matching features of **awk**, you need to be familiar with the metacharacters used in regular expressions (see section 5.4.5). A pattern is enclosed between forward slashes (**/**) on the command line or in a script. When **awk** reads an input line, it attempts to match each pattern-matching rule in a script. Only the lines matching the particular pattern are the object of an action. If no action is specified, the line that matches the pattern is printed (executing the **print** statement is the default action).

In our `personnel` example, let's assume that Alice is from Illinois (IL) and Eric is from Iowa (IA). To bring up the complete records with the pattern IL or IA, we could issue the command:

```
% awk '/I./{print}' personnel
```

where the metacharacter **.** matches any single character. We could more simply type:

```
% awk '/I./' personnel
```

and get the same result in either case:

```
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Eric Smith 24 Birch St., Albert City, IA 50510
```

In Appendix D we present **awk**'s programming model, which is beyond the scope of the present section. This model will help you understand the potential that **awk** offers the programmer.

sort

sort sorts the lines of the specified files, typically in alphabetical order. Using the **-m** option it can merge sorted input files. Its syntax is:

% sort [*options*] [*field-specifier*] [*filename(s)*]

For example, start with the `personnel` file contents:

```
John Smith 75 South Ave., Denver, CO 80145
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Mary Fahey 901 California St., San Francisco, CA 94121
Eric Smith 24 Birch St., Albert City, IA 50510
```

Run the command:

```
% sort personnel
```

to reorder the file contents as follows:

```
Alice Jones 834 S. Jefferson St., Batavia, IL 60510
Eric Smith 24 Birch St., Albert City, IA 50510
John Smith 75 South Ave., Denver, CO 80145
Mary Fahey 901 California St., San Francisco, CA 94121
```

sort is very easy to use. Read the man page for **sort** to see what the available options are and how to specify the sort fields. If a field is not specified, the sort key is the entire line. The sorted output goes to standard output by default.

5.4.5 Regular Expressions

A *regular expression* is a string composed of letters, numbers, and special symbols that defines one or more strings. They are used to specify text patterns for searching. This is similar to wildcards on VMS.

A regular expression is said to *match* any string it defines. The major capabilities include:

1) match single characters or strings of characters

2) match any arbitrary character

3) match classes of characters

4) match specified patterns only at the start or end of a line

5) match alternative patterns

Regular expressions are used by **vi**, **grep**, and **awk** (and at least a couple of utilities not covered in this manual, for instance **ed** and **sed**). **grep** in fact stands for **g**lobal **r**egular **e**xpression **p**rinter. For a complete discussion of regular expressions, refer to a UNIX text. To get you started, we include a table of special characters that can be used in expressions.

Note that regular expression special characters are different from those used in filename expansion.

`.`	Matches any single character Example: *.ing* matches all strings with any character preceding *ing*; sing*ing*, *p*ing
`*`	Represents 0 or more occurrences of the preceding character Example: *ab*c* matches *a* followed by 0 or more *b*'s followed by *c*; *ac*, *abc*, *abbbbbc*
`.*`	Matches any string of characters (`.` matches any character, `*` matches any number of occurrences of the preceding regular expression)
`$`	Placed at the end of a regular expression, matches the end of a line Example: *ay$* matches *ay* at the end of a line; ... tod*ay*
`^`	Placed at the beginning of a regular expression, matches the beginning of a line Example: ^*T* matches a *T* at the beginning of a line; *T*oday ...
`"`	Delimits operator characters to prevent interpretation
`\`	Turns off special meaning of the following single character (\ is often called a *quote* character)
`[]`	Specifies character classes
`[...]`	Matches any one of the characters enclosed in square brackets Example: *[bB]ill* matches *bill* or *Bill*

There is an extended set of special characters available for full regular expressions, including for example `?` and `+`. These can be used in **egrep** and **awk**. Refer to a UNIX book for information.

5.5 Job Control

Any command you give to the shell (true for all shells except **sh**) is a *job* and is given a *job number.* A single command is the simplest job. A series of commands separated by semicolons, or commands piped together, create a single job. A script also creates a single job. A job may consist of many processes, because each command is a process.

The job stays with its environment, for example, the current directory. If you subsequently change directories after putting a job in the background and then resume the background job, you will be in the original directory again.

Job control allows you to work on several jobs at once, switching back and forth between them at will, and it allows you to stop, start, and kill them. When you start up a job interactively, it is by default in the *foreground* and attached to your terminal. You can move that job into the *background* so you can start up another job or observe another job that is already running. You can move any background job into the foreground so it is once again attached to your terminal. You can run any number of background jobs at any one time, but there can be only one foreground job. The use of multiple windows on an X terminal makes much of this transparent.

5.5.1 Priority

You can control the priority of a command or shell with the shell command **nice**:

% nice [+*n* | -*n*] [*command*]

n is the value by which you want to increase or decrease priority. Values range from 1 to 19, with the default at 10. The higher the **nice** value, the lower the priority of a process, and the slower it runs. (You are being *nicer* to other users!) If no number is specified, **nice** sets the priority to 4. If ***command*** is omitted, the priority is set for the current shell. If ***command*** is specified, it is run at the specified (or default) priority in a sub-shell. You can use **nice** to lower the priority of a command or shell that makes large demands on the system but isn't needed right away.

Note that another **nice** command exists, `/bin/nice`. It is not a built-in shell command. If you do **man nice**, you will get information on this one. In order to get information on the shell command **nice**, do **man csh** (using **csh** as an example).

5.5.2 Background, Foreground, and Suspended Jobs

You run jobs in the *background* so that you can perform other tasks in the foreground (i.e., interactively). Jobs are always in one of three states: running in the foreground, running in the background, or suspended. Any job intended to run in the background should have its output and error redirected to a file.

There are two ways to put jobs into the background:

Using the & Metacharacter

One way to start a job in the background is to append the ampersand metacharacter (**&**) to the end of the command line. In the first example, the standard output is redirected to a file (in this case, the syntax is valid for both shell families):

% *command* > *output-file* &

Note that the parentheses are necessary in the next example in order to send both commands to the background:

```
% (command1 ; command2) &
```

The shell prints a line indicating the job number and process ID of its top-level commands, and the job is started as a background job.

Using the Suspend Control Character

The other way is to use the suspend control character, called *susp* or *swtch*, (see section 2.4) which is usually assigned to <Ctrl-z>. It stops or *suspends* the foreground (the currently running interactive) job, moving it to the background; it does not kill it.

After stopping a job, you can either resume it with the **fg** command or make it run in the background with the **bg** command (see below). You may want to stop a job temporarily to do another task and then return to it interactively, or you may want to stop it in order to let it finish as a background job.

When a background job terminates, this is reported just before the next prompt (so the message doesn't interrupt the current foreground job).

A background job will stop if it tries to read from the terminal. If output is not redirected, a background job can either (continue to) send output to the terminal or be stopped if it attempts to write to the terminal. The following command can be used to toggle this behavior:

```
% stty [-]tostop
```

The minus indicates negation, meaning that background jobs will continue to run even if they attempt to write output to the terminal and that the output will appear on the terminal screen. However, programs which attempt to interrogate or change the mode of the terminal will be blocked when they are not in the foreground whether or not *tostop* is set.

Listing Jobs

The **jobs** command lists your jobs:

```
% jobs [-l]
```

This command lists the background jobs, their job number, status, and the command being executed. A plus sign in the output means that job is *current* (in control of your terminal), a minus sign means that job is *next*. *Current* and *next* refer to its relation to the foreground (see **fg**). The **-l** option lists the process ID as well.

Commands Used for Controlling Jobs

There are a number of commands to control jobs: **fg**, **bg**, **stop**, **kill**. All of them can take an argument which specifies the particular job, or they can have no argument. The argument can take two basic forms: a simple process ID number (as displayed by **ps**) or a form prefixed with a percent sign (**%**). If no argument is given, the current job is acted upon.

The **%** form of the argument can be **%-** where **-** indicates the previous job, ***%n*** where ***n*** is the job number as displayed by the **jobs** command, ***%pref*** where ***pref*** is some unique prefix of the command name and arguments of one of the jobs, or ***%?str*** where ***str*** is some unique string in one of the jobs.

You can use the **fg** command to move a suspended or background job into the foreground:

```
% [fg] %[job]
```

The **fg** is not mandatory. If the job specification is omitted, the current job will be brought into the foreground, and the next job becomes current.

Examples (note that the first `%` on each line represents the default **csh** prompt):

`% fg %5`	Bring job number 5 into the foreground
`% %1`	Bring job number 1 into the foreground
`% %`	Bring the current job into the foreground

After stopping a foreground job, you can start it running in the background with the `bg` command. `bg` puts the current or specified jobs into the background, continuing them if they were stopped. In the following commands, ***job*** stands for job number.

`% bg %[`***job***`]`

We described above how to stop (suspend) a foreground job with the suspend control character (<Ctrl-z>). Similarly, you can suspend a background job with the `stop` command:

`% stop %`***job***

You can abort a suspended or background job with the `kill` command:

`% kill %`***job***

If you attempt to exit a shell (logout) when there are stopped jobs, you will get a warning message. A second `logout` will log you out if you choose not to see what jobs are stopped before you exit. In the C shell family, background jobs will continue running after you log out.

5.5.3 Scheduling Jobs: at and cron

UNIX provides two methods for running jobs at some specified time.

If AFS is installed on your system, there are Kerberos authentication problems with running programs that spawn jobs external to your login process group (Kerberos authentication is described in section 7.3). **at** and **cron** fall into this category. You can run the job, but it will not run with authentication, and most likely will not be able to write into `/afs` space. A work-around is available for executing an authenticated **cron** job (send mail to *helpdesk@fnal* requesting the full details of this procedure).

at

The first is the **at** utililty. This allows the user to queue a job for later execution.

The format of the `at` command is:

`% at` ***time*** `[`***date***`] [+`***increment***`]`

at reads the commands from standard input. Standard output and standard error output will be mailed to you unless they are redirected.

The shell saves the environment variables and the working directory that are in effect at the time you submit the job and makes them available when the job is executed.

- The ***time*** can include 1, 2, or 4 numbers. One or two digits is assumed to be hours, four digits to be hours and minutes. It can be specified as two numbers separated by a colon (hours:minutes), either in 24-hour format or with *am* or *pm* appended. The names *noon*, *midnight*, *now*, and *next* are recognized.
- The ***date*** is either a month name followed by a day number (and optionally a year number followed by an optional comma) or a day of the week (fully spelled out or abbreviated to three characters). The words *today* and *tomorrow* are known. If no date is given, *today* is assumed if the hour is greater than the current hour and *tomorrow* if it is less.

- The optional ***increment*** is a number suffixed by *minutes*, *hours*, *days*, *weeks*, or *years* in singular or plural form.

Examples:

```
% at 8

% at 0800

% at 8:00am Jan 24

% at now + 1 minute
```

at reads from standard input, meaning you type in the commands (there may or may not be a prompt). When you are finished, terminate input with <Ctrl-d> followed by a carriage return.

You can also redirect the input to a file of commands, for example:

```
% at now + 1 hour < myscript
```

at runs in the Bourne shell (**sh**) by default. If you need to force it to run in C shell, you can use the trick illustrated in the following interactive example:

```
% /bin/csh << xxxxx

? at now + 2 minutes

? source .cshrc

? alias > aout

? <Ctrl-d> (followed by carriage return)
```

The first line causes **csh** (C shell) to read the following lines up to **xxxxx** or to the end-of-file. There is no **xxxxx**, of course, so it reads until you give it the <Ctrl-d>. The third line runs your `.cshrc`. It is an illegal Bourne shell command, therefore you can tell **at** ran in the C shell and that your `.cshrc` file was executed. You will receive a message similar to the following, and the results will be mailed to you (alas, **at** will say it's using `/bin/sh` even if you've "tricked" it):

```
warning: commands will be executed using /bin/sh
job 826157640.a at Wed Mar  6 18:14:00 1996
```

After 2 minutes, `aout` is mailed to you. It contains a list of all the aliases defined in your `.cshrc` file. If you are running a script using **at** then the script will be run under whatever shell you specify in the script.

For example, say you run:

```
% at now + 2 minutes

? script

? <Ctrl-d> (followed by carriage return)
```

where ***script*** is a file that contains the line `#!/bin/csh` at the beginning. The commands in the script will execute under **csh**.

cron

The second method for running jobs at some specified time is the **crontab** command. It is designed for jobs that need to be run on a regular basis, e.g., once a night, or once per week. Note that **cron**, like **at**, uses the Bourne shell so that output redirection must be specified using Bourne shell syntax. Scripts will be run under what ever shell is specified in the script. If no shell is specified then Bourne shell is used.

The format of the command is:

% crontab [*filename*]

% crontab [*options*]

where ***filename*** is the name of a file containing the commands that you want to have executed. If you do not specify a file, then **crontab** will read commands from standard input as you type them, ending with <Ctrl-d>, and the commands will be run in Bourne shell. The system utility **cron** reads the crontab file and runs the commands. Standard output and standard error will be mailed to you unless they are redirected.

The command can also take the following options:

-r	remove crontab file
-l	list contents of crontab file

A **cron** file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

- minute (0-59)
- hour (0-23)
- day of the month (1-31)
- month of the year (1-12)
- day of the week (0-6 with 0=Sunday)

If an asterisk appears in a field instead of a number, **cron** interprets that as a wildcard for all possible values. The sixth field of a line in a **cron** file is a string that is executed by the shell at the specified times.

Examples:

The user creates a **cron** file `myfile`, and runs **crontab**:

```
#Myfile
# Run script that archives to 8mm tape for backup.
# Monday-Thursday at 2200 backup everything that has been
# changed.  Every Friday at 2200 backup everything.
0 22 * * 1-4 /usr/buckley/daily  1>>/usr/buckley/cron/backup.log 2>&1
0 22 * * 5   /usr/buckley/weekly 1>>/usr/buckley/cron/backup.log 2>&1
```

% crontab myfile

This command will perform an incremental backup at 10pm Monday to Thursday and a full backup at 10pm on Friday.

Chapter 6: The UNIX File System

The UNIX file system has a hierarchical or tree-like structure with the directory called *root* (`/`) as its source. The system is essentially composed of *files* and *directories*. In this chapter we describe techniques for manipulating files and directories, and commands designed to provide information about them.

In Appendix E you can find a table of UNIX equivalents for commonly used VMS commands.

6.1 Directory Structure

The UNIX system automatically puts you at a specific location in the file system when you log in. This is called your *login directory*. Typically, this is the same as your *home directory*. The name of your home directory is usually the same as your login name. Within this directory you can create files and additional directories (sometimes called *subdirectories*) in which to group the files. You can move and delete your own files and directories and control access to them.

The root of the file system is called *root* and is written as a slash (`/`). In other words, to change to the root directory, type:

```
% cd /
```

There is only one directory tree on a system even if several devices are mounted in that tree. (All devices are viewed as files.) The *current directory* or *working directory* is the directory that you are currently working in, which is also the directory that commands refer to by default. Files in your current directory can, therefore, be specified by their filenames only.

6.1.1 Pathnames

Wherever you can use a filename, you can also use a *pathname*, which is how you point to files that are not in your current directory. You can refer to files in other directories using either a *relative path name*, that is a path specified relative to your current directory, or with an *absolute path name*, that is a path specified relative to the root of the file system.

Absolute path names are preceded with `/`, the root directory. If a pathname does not begin with `/` it is assumed to be a relative path name. Relative path names begin with a directory or filename, a `.` (pronounced "dot") which refers to the current directory, or `..` (pronounced "dot dot") which refers to the directory immediately above the current directory. The character `/` also separates components of the pathname, which are directory names, except for the last one, which can be either a simple filename or a directory name.

In summary, every file has a pathname, and its absolute pathname is of the form:

```
/rootdir/dir2/... /filename
```

The following is the form of a relative pathname of a file:

```
dir_n/dir_n+1/... /filename
```

An example of an absolute path name is:

```
/usr/smith/project1/afile
```

If my current directory is `/usr/smith`, then I can refer to the file `afile` in subdirectory `project1` with a relative pathname like this:

```
project1/afile
```

Or, if my current directory is `/usr/smith/project1`, I can refer to a file named `fileb` in `/usr/smith/project2` as:

```
../project2/fileb
```

Note that you cannot necessarily tell if `fileb` is an ordinary file or a directory name. Many commands will accept a directory name, and if it *is* a directory name, the command in which it is used may perform the action on all files in the directory. This behavior can be dangerous!

6.1.2 The Home Directory

Your home directory is the top of your personal branch in the file system, and is usually designated by your username, i.e. `/{path}/{username}`.

Tilde (~)

In most UNIX shells other than **sh**, the *tilde* (~) stands for the home directory. Used alone, it specifies your home directory. Followed by a different user's login name, it expands into the pathname of the home directory of that user. This is a convenient way to refer to a user's directory, because it is independent of where the system manager may place the directory on the disk.

The use of tilde (~) to refer to a home directory is limited. It isn't available in the Bourne shell, and isn't available in FORTRAN.

Of the following three examples, the first refers to the file `def` from your own home directory, the second to the home directory of user *jones*, and the third refers to file `data1` in the subdirectory `project1` of jones' home directory.

```
~/def
~jones
~jones/project1/data1
```

To change to jones' home directory you'd enter:

```
% cd ~jones
```

logdir

FUE provides the command **`logdir`** which returns the full path of the specified user. **`logdir`** by itself returns the path of the invoking user. For example:

`% logdir` ***username***

is equivalent to:

`% echo` ***~username***

To change directories you'd enter (note the use of backquotes to use the output of the enclosed string):

```
% cd `logdir username`
```

In contrast to the tilde, `logdir` can be used within commands, scripts, FORTRAN and C programs, and other programs in all shells.

$HOME

The environment variable *HOME* is automatically set to the absolute pathname of your home directory. Environment variables are discussed in section 9.1 and *HOME* is described in section 9.2. To see the value of *HOME*, enter:

```
% echo $HOME
```

From some other directory, you can change to your home directory or one of its subdirectories using a command like the following:

```
% cd $HOME[1]
```

or

```
% cd $HOME/mysubdir
```

6.1.3 Command Line Directory Shortcuts

.	Current directory
..	Parent directory of the current directory ("up" one directory)
~	Your home directory (all shells but **sh**)
$HOME	Environment variable whose value is your home directory
~username	Home directory of another user (all shells but **sh**)
/	Root directory

6.1.4 Directories and Executables

It is appropriate at this point to mention the relationship between directories and commands. A command is simply the name of an executable file, located in some directory. To execute a command, the shell first needs to find the executable file. The shell therefore needs to be given a set of directories to search. This information is provided via the environment variable *PATH* which is a list of search directories. You can display it with the command:

```
% echo $PATH
```

PATH is explained more thoroughly in section 9.2. Standard UNIX commands are generally grouped in a few standard directories (e.g., `/usr/bin`), and your default *PATH* contains these. See section 9.2 to learn how to run executables that you create and store in your own directories.

---

1. `cd` by itself is equivalent to `cd $HOME`.

The utility **which** is useful in cases where a command may be ambiguous, for example due to aliasing (see section 9.3), and you want to know exactly which executable file or files the command runs. **which** lists the files that would be executed if the specified command(s) had been run. The syntax for **which** is:

% which *command* [*command2* ...]

Each argument is expanded if it is aliased, and your path is searched for the executable files associated with the commands. See the man page for more information.

6.2 Files

An ordinary file contains ASCII characters or binary data and is considered by the UNIX system to be merely a sequence of bytes. No structure is imposed on the file and no meaning attached to its contents by the system; the meaning depends on the program that reads the file.

A directory file contains an entry for each file in that directory. The directory entry for a particular file contains the file name and *inode number.* The inode number is a volume data structure used by the file system. It has an associated entry in the *inode table* which contains other information about the file such as the owner, file protection, modification date.

A *hidden file* is an ordinary file whose name begins with a period (called "dot"). The `.login`, `.cshrc`, and `.logout` files described in Chapter 9 are hidden files. The reason they are called hidden is that the **ls** (list files) command does not list them by default. Use the **-a** option with **ls** to see them. Hidden files do not appear in filename expansion of *, either. Filename expansion is discussed in section 6.2.2, below.

UNIX does not support file versions. If you edit a file and save it with the same name, your earlier version is overwritten. Similarly, if you copy or rename (move) a file to a filename that already exists, the original file is overwritten.

6.2.1 Filenames

A full file specification has only two parts, the directory specification and the file name. A filename is composed of from 1 to 14 characters in old UNIX implementations and a much larger number in more recent versions (up to 255, typically). Although you can use any character in a filename except **/**, UNIX assigns special meaning to many characters (metacharacters), so they should be avoided (see section 2.5). It is safe to use the upper- and lowercase letters, numbers, dash (-), underscore (_), period (.), and comma (,). As mentioned in the previous section, files beginning with a dot (.) are hidden files. The "filenames" **.** and **..** (single and double dot) are reserved. The **.** refers to the current directory, and the **..** refers to the current directory's parent directory. No two files in the same directory can have the same name, but files in different directories can have the same name.

VMS users have not had the luxury of using dashes in filenames. Dashes are much more common in UNIX filenames simply because it's easier to type `my-file` than `my_file`.

Filenames are case sensitive. This means `MYFILE` is different from `Myfile` is different from `myfile` is different from `myFile`, etc.

You cannot distinguish a directory file from an ordinary file by its name, although some people make their own convention by beginning directory filenames with a capital letter, or ending them in `.d`.

Filename extensions are not required in UNIX. You can include a period and an extension in a filename to help describe the contents of the file, but it will not have special meaning to UNIX itself. However, programs can make use of extensions, for example the FORTRAN compiler expects certain extensions. Note, you can have more than one period in a filename, for example, `lex.yy.c`.

6.2.2 Filename Expansion and Wildcard Characters

The UNIX shells have a number of special characters which can be used on the command line when specifying filenames and directory names. They allow the shell to expand the argument into a set of filenames. These characters are called *wildcards*. Filename references that contain these characters are called *ambiguous file references*. Filename expansion is also called *globbing*.

The question mark (**?**) causes the shell to generate filenames which match any single character in that position. For example, `out?` matches `out1` but not `out12`.

The asterisk (*****) causes the shell to generate filenames which match any number of characters (including zero characters) in that position. For example, `myfile` matches `myf*`. The * alone means all files (except those that begin with dot (**.**), which is a special case).

A pair of brackets (**[ ]**) surrounding a list of characters causes the shell to match filenames containing the individual characters in that position. The brackets define a *character class* and each definition can only replace a single character in a filename. In other words, it is like a question mark that will only allow certain characters. For example, `memo1` and `memoa` match `memo[14a]`), but `memo3` and `memo1a` do not. A hyphen can be used to define a range of characters, for example `[a-z]` represents all lowercase characters. Thus `memo[a-z]` matches `memoa` but not `memo2` or `memoB`.

Character	Action
?	matches any single character in a filename
*****	matches any string of characters (including the empty string) in a filename
[ ]	matches any single character from the set enclosed in the brackets

Examples:

% ls out*	lists all files beginning with `out`
% ls out?	lists all files with 4-character names beginning with `out`
% ls out[ab]*	lists all files beginning with `out` followed by `a` or `b` (e.g., `outa4`)
% ls *out*	lists all files containing `out`

Filename expansion may surprise you with the results. For example, **`ls b*`** would list all files starting with b in the current directory, but it would also list the contents of all **directories** whose names start with `b` because of the way **`ls`** behaves for a directory argument. If you want to be

sure of what filename expansion will result in, you can use the **echo** command to check it before executing a command.[1] For example, say you have a few matching files in your directory for the command:

```
% echo *out*
```

You would obtain output something like this:

```
fout fout275 inandout out1 out2 out
```

Filename expansion in **csh** can be turned off by setting the *noglob* variable:

```
% set noglob
```

To turn it back on, type **unset noglob**.

6.3 Manipulating Files

This section describes the basic file manipulation commands:

- listing the contents of a directory
- displaying the contents of a file
- copying and renaming a file
- deleting a file
- changing a file's access permissions

Section 6.5 describes the commands you can use to change and manipulate directories.

6.3.1 List Directory Contents: ls

The **ls** command, which stands for **list**, is used to list the contents of a directory. **ls** has many options, some of which are system-dependent, so only a few of them are described here. For a complete description of the command, refer to the man pages for **ls**.

ls by itself lists the names of the files and subdirectories in the current directory (in multicolumn format on some platforms), sorted alphabetically.

The format is:

% ls [*options*] [*filenames*]

where some of the options are:

-a	List all entries, including those that begin with **.** (dot).
-l	List in the long format, giving mode, number of links, owner, group, size in bytes, and (by default) time of last modification, by default sorted by filename.
-C	List in columns (default on some platforms)
-F	Put a / after the name of each file that is a directory, an * after the name of each file that is executable, and an @ after the name of each file that is a symbolic link.

1. **echo** is otherwise useful for sending messages to the terminal from a script and sending known data into a pipe.

`-R`	Recursively list subdirectories encountered.
`-t`	Sort by time stamp (latest first) instead of by name. The default time stamp is the last modification time (see `-u`).
`-u`	Use the time of last access for sorting if used with the `-t` option or printed in the date column if used with the `-l` option. `-ult` both sorts by and displays last access date.
`-d`	If the argument is a directory, list the directory itself, not its contents. Use with `-l` to get the status (e.g., permissions) of a directory.

If the argument is a directory, `ls` displays the contents of the directory. Note that this can happen unintentionally as a result of filename expansion. This behavior can be prevented with the `-d` option. The `-t` option is useful when looking at recent files:

```
% ls -lt
```

will result in the long output sorted by reverse modification date rather than by filename.

The following is a sample output of `ls -l`.

```
total 251
drwxr-xr-x 3 nicholls g020c 512   May 2  08:53 Tools
drwxr-xr-x 2 nicholls g020c 512   May 2  09:01 bin
-rw-r--r-- 1 nicholls g020c 446   May 4  14:09 defaults
-rw-r--r-- 1 nicholls g020c 95418 May 1  17:42 intro.lpr
-rw-r--r-- 1 nicholls g020c 0     May 10 17:51 lsout
-rw-r--r-- 1 nicholls g020c 6683  May 1  16:46 man.lpr
-rw-r--r-- 1 nicholls g020c 12258 May 9  16:16 out
```

The first line indicates the number of blocks used. The rest of the lines report on (sub)directories or files in the directory being reported on. The first column of the output is called the *mode*. The character in this first column indicates the type of file, and for our purposes here, they are:

`d`	directory
`-`	ordinary file

The next 9 characters are interpreted as three fields of three characters each, indicating the read (`r`), write (`w`), and execute (`x`) permissions for *owner* (sometimes called *user*), *group*, and *other*, in that order (see section 6.6.1 for a discussion of permissions).

Next is the number of links to the file or directory. This refers to the number of different names established for it. Normally files have 1, and directories have 1 each for the directory itself, its parent directory, and each of its subdirectories. In the sample output above, notice that the directory `Tools` has 3 and `bin` has 2. Evidently, `Tools` has one subdirectory and `bin` has none.

The next fields are the login name of the owner, the group to which the owner belongs, the size of the file in bytes, the date and time the file was last modified, and, finally, the filename (which can be a directory name).

6.3.2 List File Contents: cat, less, more, head, and tail

UNIX has a number of commands that can be used for displaying the contents of a file at the terminal.

cat

cat, which stands for "con**cat**enate and print," is the standard UNIX file display; it simply prints the file to the screen.[1] When piped to **less** (see section 5.4.4 which describes **less** as a filter), **cat** displays the file contents a screen at a time, and some simple commands may be executed at the supplied prompt.

```
% cat filename... [| less]
```

As its name suggests, **cat** is in fact quite useful for copying and concatenating files. Output is often piped to a file rather than to the screen, using standard output redirection (see section 5.4.2). The following example concatenates the three specified files and copies them sequentially to a single file called `allthreefiles`:

```
% cat fileone filetwo filethree > allthreefiles
```

less, more

A shortcut for **cat** *filename* **|less** is to use **less** as a file browser:

```
% less filename
```

And wherever you use **less**, you can alternatively use **more**, although it is not as functional as **less**. You cannot move backwards through the file with **more**.

head, tail

head displays the first *n* lines of the specified file or files. If more than one file is specified, the filename is displayed before each set of file contents of *n* lines. *n* defaults to 10 lines.

```
% head [-n] [filename...]
```

tail displays the last lines of a file. Its syntax is a bit different:

```
% tail [+|-n lbc] [filename...]
```

The option *+n* displays the file starting *n* lines down from the beginning of the file, *-n* displays the last *n* lines. **l**, **b**, or **c** requests display of *n* lines, blocks, or characters (default is **l** lines). If more than one file is specified, the filename is displayed before each set of file contents. *n* defaults to 10 lines.

tail is useful when you want to see how far a process got. To display the last line of a log file, enter:

```
% tail -1 logfile
```

6.3.3 Copy a File: cp

The command **cp** (stands for **cp**y) can be used to make a copy of a file, leaving the original version intact. You can copy a single file to another one (in the same or a different directory), or you can copy one or more files to a different directory, retaining the same filenames.

---

1. There are better ways to display a file (see **less** and **more**, which follow **cat** in this section).

The syntax for these two situations varies slightly:

% cp [*options*] *file1 targetfile*	the file ***file1*** is copied to ***targetfile***, where ***targetfile*** may include a path
% cp [*options*] *file1* [*file2 ...*] *targetdirectory*	one or more files (***file1 file2*** ...) are copied to ***targetdirectory***

If the target is a file, its contents are overwritten unless **-i** is specified, in which case you are prompted for confirmation.

Some options are:

-i	If the target filename exists, you are prompted for confirmation before overwriting.
-r	Used only with the ***targetdirectory*** form. Recursively copy a directory, its files, and its subdirectories to ***targetdirectory***.

The first example below copies `myfile` to `anotherfile`, both in my current directory, prompting for verification in case `anotherfile` already exists.:

```
% cp -i myfile anotherfile
```

New users may find it useful to define **cpi** as the alias for **cp -i** to use in place of **cp** so that prompting always occurs. Section 9.3 discusses aliases.

The second example copies files `proj1` and `proj2` to another directory named `newproj` which is parallel to the current directory (has same parent directory as current):

```
% cp proj1 proj2 ../newproj
```

The third example copies the file `oldproj/proj1` to my current directory (`.`), which is a parallel directory to `oldproj` (has same parent directory). The file `proj1` keeps its name.

```
% cp ../oldproj/proj1 .
```

6.3.4 Move (Rename) a File: mv

The **mv** command (stands for **mo**ve) allows you to rename a file in the same directory or move a file from one directory to another. If you move a file to a different directory, the file can be renamed or it can retain its original name. **mv** can also be used to move and rename directories.

% mv [*options*] *source1* [*source2 ...*] *target*

Depending on whether the ***source(s)*** and ***target*** are files or directories, different actions are taken. These are described in the table below. If ***target*** is a filename, only one ***source*** file may be specified.

Source	Target	Result
file	***name***	Rename file as ***name***
file	existing file	Overwrite existing file with source file
directory	***name***	Rename directory as ***name***

Source	Target	Result
directory	existing directory	Move directory to be a subdirectory of existing directory
one or more files	existing directory	Move files to directory

An important option is:

`-i` If ***target*** exists, the user is prompted for confirmation before overwriting.

6.3.5 Reference a file: ln

The `ln` (link) command allows you to create a link in one directory to a file in the same or a different directory, or to a different directory. Via links, a file or directory can appear to exist in multiple places, but only actually exist in one, thus conserving disk space. Links are often used to easily reference files or directories that would otherwise require a long path name.

The syntax for `ln` is similar to that for `cp` and `mv`, and in fact they are all run by the same executable.

The most commonly used options for the `ln` command are:

`-i` You are prompted before overwriting an existing filename.

`-s` This makes a *symbolic*, as opposed to an ordinary or *hard*, link. A symbolic link can point to a file that is in a different file system, whereas a hard link cannot.

A symbolic link displays the link and the file to which it is linked when you run `ls -l`; this is the only way to know the name that a file is linked to.

Note that when using the AFS file system, hard links can only be made between files that are in the same directory (the same *volume*, see section 7.5), so use the `-s` option even if you're in the same directory tree.

The syntax differs slightly for files and directories:

`% ln [`***options***`] /path/to/file_name /path/to/link_name`

Create the link ***link_name*** to reference the file ***file_name***. If ***link_name*** already exists (as a link or as a file), it gets overwritten (unless you use option `-i`).

`% ln -s [`***other options***`] /path/to/file_name /path/to/link_name`

Create symbolic link named ***link_name***, that links to ***file_name*** which exists in the same or another directory.

`% ln -s [`***other options***`] /path/to/file_name1 [/path/to/file_name2 ...] directory`

Create a symbolic link in ***directory*** to each of the listed files. The files may all exist in different directories since the `-s` option is used. The link names will be the same as the filenames they link to. If files of the same name but in different directories are specified, only the first link of that name will be created.

Let's look at an example:

```
% ln -s /e741/run1/e_mu2/mydata r5742
```

If `r5742` is a directory, this creates a link called `mydata` in the directory `r5742` that points to `/e741/run1/e_mu2/mydata`. You can now reference the data file as `mydata` (i.e. the same filename) as if it were in the directory `r5742`.

On the other hand, if `r5742` is not an existing directory then it represents the name of the link being created. In this case, the command establishes a link called `r5742` that points to the file `/e741/run1/e_mu2/mydata`. Running the command **`ls -l`** should display the following output:

```
lrw-r--r--   1 aheavey  g020          46 Aug 29 14:26 r5742 -> /e741/run1/e_mu2/mydata
```

6.3.6 Remove a File: rm

The **`rm`** command (stands for **r**emove) is used to remove the entries of one of more files.

`% rm` **[*options*]** ***file...***

Some commonly used options are:

`-i` Confirmation of removal of write-protected file occurs interactively, whether the standard input is a terminal or not. If used with the **`-r`** option, you are prompted about each directory before it is examined.

`-r` Causes **`rm`** to delete the contents of the specified directory, including all its subdirectories, and the directory itself (recursive). This option should be used cautiously.

The file list can include ambiguous file references, so **`rm`** should be used cautiously. You can use the **echo** utility with the same ambiguous file reference to see the list generated.

Removal of a file requires write permission to its directory, but neither read nor write permission to the file itself. If the file has no write permission and the standard input is a terminal, the set of permissions is printed and you are prompted for confirmation. If the answer begins with a **y**, the file is deleted. If the standard input is not the terminal, the files are deleted without confirmation.

New users may find it useful to define **`rmi`** as the alias for **`rm -i`** to use in place of **`rm`** so that prompting always occurs. Section 9.3 discusses aliases.

6.3.7 Copy to/Restore from Archive or Tape: tar

The **tar** utility (**t**ape **ar**chive) can be used to create, add to, list and retrieve files from an archive file. Archive files are often stored on tape. The action taken by the **`tar`** command depends on the *key*, which is essentially a function option. The key must be specified on the command line as if it were the first option. It may be followed by function modifiers, and then by options and/or arguments. The keys and function modifiers must be grouped together before any arguments are listed. **tar** does not require, but does allow, a dash (**-**) before the list of keys and function modifiers. The keys and functions are:

`c` create a new tar file

`r` append specified files to tar file

`t` list all files in the tar file, or all files in a specified file list

`u` append new or changed files to tar file

`x` unwind entire tar file or extract specified files from tar file and write each file to the directory as specified in the tar file relative to the current directory

The keys, function modifiers and options are discussed in the man pages. Be aware that they vary in some cases between UNIX flavors. The command syntax varies somewhat from key to key, so check the man pages for that information, too.

When creating a tar file, we have a few recommendations for avoiding problems:

First, when possible, create the tar file on a machine of the same flavor as the target platform. Occasionally a tar file doesn't unwind properly on a different platform.

Secondly, choose your working directory carefully. It is often convenient or desirable to be able to specify simple relative path names for the files to include in the tar file. For example:

```
% cd /path/to/dir

% tar cvf /tmp/filename.tar .
```

creates a tar file with all pathnames relative to `/path/to/dir`. In general you should not specify the pathname explicitly on the command line, unless it will be valid on any other system where the tar file may be unwound and used.

Thirdly, be careful choosing your target directory for the new tar file. Make sure that the target directory is outside of the directory tree that you're including in the tar file. Otherwise the tar file tries to include itself, and can grow infinitely large.

Note to ex-VMS users: **tar** does not provide for extracting a named set of files from a backup file and placing them all in the current directory disregarding their directory specifications on the backup file.

6.3.8 Compress or Expand a File: gzip, gunzip

Several utilities are available on UNIX systems for file compression. **compress** and **pack** are native UNIX utilities, and **gzip** is provided by FUE. We recommend you use **gzip** for file compression, and its associated utility **gunzip** for file expansion. **gunzip** recognizes and can expand files compressed with **compress** and **pack** as well as **gzip**.[1]

The file extensions **gunzip** recognizes include `.gz`, `-gz`, `.z`, `-z`, `_z`, and `.Z`. **gunzip** also recognizes the special extensions `.tgz` and `.taz` as shorthands for `.tar.gz` and `.tar.Z`, respectively. When compressing, **gzip** uses the `.tgz` extension if necessary instead of truncating a file with a `.tar` extension.

You will need to reference the man pages for details on syntax, options and usage. In their simplest forms **gzip** and **gunzip** can be used as follows, starting, for example with the original uncompressed file `bigfile`:

```
% gzip bigfile
```

The result is `bigfile.gz`, whose size is reduced with respect to `bigfile` according to Lempel-Ziv coding (LZ77), the same compression scheme used by **compress**. Whenever possible, **gzip** replaces each file by one with the extension `.gz`, while keeping the same ownership modes and access and modification times. **gzip** will only attempt to compress regular files. In particular, it will ignore symbolic links. If the compressed file name is too long for its file system, **gzip** truncates it.

Compressed files can be restored to their original form using **gunzip**, or equivalently by using the **`-d`** option with **gzip**. If the original name saved in the compressed file is not suitable for its file system, a new name is constructed from the original one to make it "legal".

---

1. On some systems (namely where **gzip** and **gunzip** are not installed in `/usr/local/bin`) you will need to run **`setup gtools`** in order to access them.

To restore `bigfile.gz` to its original name and size, enter:

```
% gunzip bigfile.gz
```

6.4 Information About Files

This section gives a cursory overview of simple uses for two very powerful commands for dealing with files: **`find`** for searching for files and **`grep`** for searching for strings within files. We also describe **`wc`** which displays the size of a file, **`od`** which creates a dump of a file, and **`file`** which can determine file type.

6.4.1 Find a File: find

The **`find`** utility tests each file in the given pathname list to see if it meets the criteria specified by the expression supplied. It does this by recursively descending the directory hierarchy for each path name. The format is:

`% find` ***path-name-list expression***

path-name-list can contain file expansion metacharacters. Each element in ***expression*** is a separate boolean criterion. A space separating two criteria is a logical AND operator, a **`-o`** separating the criteria is a logical OR operator. A criterion can be negated by preceding it with an exclamation point (**`!`**). Criteria are evaluated from left to right unless parentheses are used to override this. Special characters must be quoted (use \) and there must be spaces on each side of the special character pair.

Some of the criteria that can be used within ***expression*** are:

`-name` ***filename***	True if ***filename*** matches the name of the file being evaluated. Ambiguous file references can be used if enclosed in quotes.
`-type` ***filetype***	True if the type of the file is ***filetype***, where ***filetype*** is either `d` (directory) and `f` (ordinary file).
`-atime` ***n***	True if the file has been accessed in ***n*** days.
`-mtime` ***n***	True if the file has been modified in ***n*** days.
`-newer` ***filename***	True if the file has been modified more recently than ***filename*** has.
`-print`	Causes the matching path names to be displayed on the screen.
`-exec` ***command*** **`\;`**	True if ***command*** returns a zero exit status. ***command*** must be terminated with a quoted semicolon (note the \). An empty pair of braces ({}) within the command represents the filename of the file being evaluated.
`-ok` ***command*** **`\;`**	Same as **`-exec`** except the generated command line is displayed and executed only if the user responds by typing `y`.

In the previous list, **+*n*** means more than ***n***, **-*n*** means less than ***n***, ***n*** means exactly ***n***.

Note that **`find`** doesn't do anything with the found files, it doesn't even display the names, unless instructed to.

Examples:

- Search the current directory and all subdirectories for the file `lostfile`:

```
% find . -name lostfile -print
```

- List all files ending in `.html` in your `/wwwork` subdirectory:

```
% find wwwork -name '*.html' -print
```

- This command will prompt you if you want to execute **`more`** on each file that begins with the letter `d` in the current directory and all subdirectories (Enter `y` if you want the file displayed.):

```
% find . -name 'd*' -ok more {} \;
```

- List all files in the current directory that *don't* begin with `m`:

```
% find . ! -name 'm*' -print
```

- Find all files in the current directory and all subdirectories that contain the string *hello*:

```
% find . -exec grep -l "hello" {} \;
```

- Remove all files in your directory tree that are named `a.t` or have the extension of `.o` and haven't been accessed in a week:

```
% find ~\(-name a.t -o -name '*.o') -atime +7 -exec rm {} \;
```

Note that using the `find` command uses many system resources.

In particular on AFS systems, you may accidentally end up searching servers all over the world if the top of the search is at the root directory (`/`). Generally you should be careful to only search the part of the UNIX tree that interests you. Here is an example:

- Look for ***filename*** starting at the root directory (`/`), and exclude searches in the `/afs` and `/nfs` branches:

```
% find / \( -name /afs -prune \) -o \( -name /nfs -prune \) -o -name filename -print
```

6.4.2 Search for a Pattern: grep

The **`grep`** utility searches the contents of one or more files for a pattern.

The format is:

`% grep [`*`options`*`] `*`pattern`*` [`*`file`*` ...]`

Some of the options are:

`-c`	Display only a count of lines that contain the pattern.
`-i`	Ignore upper/lower case distinctions during comparisons.
`-l`	Display only the name of each file that contains one or more matches.

The pattern can be a simple string or a regular expression (see section 5.4.5). You must quote regular expressions that contain special characters, spaces, or tabs (this can be done by enclosing the entire expression within single quotation marks).

Examples:

- Find all non-hidden files in the current directory containing the string *smith*:

```
% grep -i smith *
```

- Search the file `abc` for a string beginning with *f*, followed by 0 or more *r*'s, and ending in *og* (e.g., *frog, fog, frrog*):

```
% grep 'fr*og' abc
```

- Search the file `myfile` for a line beginning with a **T**:

```
% grep '^T' myfile
```

or

```
% less myfile | grep '^T'
```

- Search `/usr/jones/junk` for the characters *file* followed by a number (e.g., *file1, file3*):

```
% grep 'file[0-9]' /usr/jones/junk
```

- Display a line if Smith is logged in:

```
% who | grep smith
```

- Show all processes being run by Smith:

```
% ps -ef | grep smith
```

- Show all environmental variables containing ***string*** in their name or their translation:

```
% env | grep string
```

- Show all aliases containing ***string*** in their name or their translation:

```
% alias | grep string
```

6.4.3 Count a File: wc

The **wc** command, which stands for word count, counts the number of lines, words, and characters there are in the named files, or in the standard input if the argument is absent. If there is more than one file, **wc** totals the count as well.

```
% wc [-lwc] [names]
```

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of **l**ines, **w**ords, and **c**haracters be reported. The default is **-lwc.**

UNIX users frequently count things by piping them into **wc**. For example, to display the number of users logged into the system, you can execute:

```
% who | wc -l
```

6.4.4 Dump a File: od

The **od** (**o**ctal **d**ump) command can be used to examine the contents of a file in various formats: octal, decimal, hexadecimal, and ASCII. The default is octal.

The format is:

```
% od [options] [file] [offset] [|less]
```

If ***file*** is not included, standard input is assumed. The options are:

-c	Produces a character dump.
-d	Produces a decimal dump.
-o	Produces an octal dump.
-x	Produces a hexadecimal dump.

The **-c** option prints non-printable characters as a printable character preceded by a backslash: **\0** is null, **\b** is backspace, **\f** is form-feed, **\n** is new-line, **\r** is return, and **\t** is tab.

The *offset* specifies where in the file the dump is to begin, if different than the beginning of the file. It is of the form `[+]`*n*`[.][b]`. The `+` is only necessary if you have no file specified so that the command interpreter knows this is the offset not the file. Without `.` or `b`, *n* indicates the dump starts at (octal) byte *n* of the file. A `.` displays *n* in decimal, a `b` in 512-byte blocks.

We recommend that you always pipe the output of `od` to `less` (or `more`) so that you can manipulate it. Large files can be unwieldy, and you may not be able to stop the output once it's going!

6.4.5 Determine File Type: file

The `file` utility can be used to determine the file type of a file according to its contents. It bases its guesses on a list of "magic numbers" recorded in a "magic file", `/etc/magic`. Some of the file types are:

- ASCII text
- C program text
- directory
- executable

`file` determines the filetype by looking at the beginning of the file and comparing it to entries in the magic file. The command format is:

`% file` ***filename...***

6.5 Manipulating Directories

This section describes the commands you can use to organize and use the UNIX directory structure. It describes how to make and remove directories, and move from one directory to another. Listing the contents of a directory (files and subdirectories) was described in a previous section. Section 6.6.2 explains the meaning of access permissions as applied to directories.

6.5.1 Print Working Directory: pwd

The `pwd` command (for **p**rint **w**orking **d**irectory) displays the path name of your working (current) directory. The command format is:

```
% pwd
```

6.5.2 List Directory Contents: ls

The `ls` command, which stands for `list`, is used to list the contents of a directory. `ls` has many options, some of which are system-dependent. A few of them are described in section 6.3.1. For a complete description of the command, refer to the man pages for `ls`.

6.5.3 Change Directory: cd

When you first log in to the system, you are placed in your home directory, which is then also your current working directory. You can use the **cd** command (for **c**hange **d**irectory) to change your current working directory. The command format is:

% cd [*directory*]

You can specify a complete path or a relative path. You can use **..** (for the parent directory) in your pathname. You must have execute permission (which provides search permission in this case) on a directory to **cd** to it.

If ***directory*** is not specified, you are returned to your home directory.

The following examples illustrate moving to different directories:

- your home directory

  % cd

- a subdirectory called `Tools`

  % cd Tools

- a colleague's home directory (using absolute pathname)

  % cd /usr/jones

- a colleague's subdirectory (using tilde)

  % cd ~jones/ourfiles

- a parallel directory (has same parent directory as current directory)

  % cd ../Tools

6.5.4 Make a Directory: mkdir

The **mkdir** command (for **m**a**k**e **dir**ectory) is used to create a directory. The command format is:

% mkdir *dirname ...*

If a pathname is not specified, the directory is created as a subdirectory of the current working directory. Directory creation requires write access to the parent directory. The owner ID and group ID of the new directory are set to those of the creating process.

Examples:

- create a subdirectory in the current working directory

  % mkdir progs

- create one with a full pathname (the directory `Tools` must already exist)

  % mkdir /usr/nicholls/Tools/Less

6.5.5 Copy a Directory

The most straightforward way of copying a directory and its contents is to pipe the output of the **ls** command (see section 6.3.1) into the file copy facility **cpio** (see the man pages).

First, create the destination directory using **mkdir** (see section 6.5.4), if it doesn't already exist. Secondly, from the source directory, run the command (shown with recommended options; see man pages for option information):

```
% ls | cpio -dumpV destination_dir
```

The ***destination_dir*** must be specified relative to the source directory.

Another way to copy directory hierarchies is to use the **tar** utility, described in section 6.3.7. The following sequence of commands copies a structure from the source directory to the destination directory:

```
% cd source_dir; tar cf - . | (cd destination_dir; tar xfBp -)
```

The "-" is used for the name of the tar file (argument to the **f** option) so that **tar** writes to the standard output or reads from the standard input, as appropriate.

6.5.6 Move (Rename) a Directory: mv or mvdir

See section 6.3.4 for information on **mv**. To move a directory (***olddirname***) and its contents to a different position in the directory tree, use the command format:

```
% mvdir olddirname newdirpath
```

If ***newdirpath*** exists already, then the directory gets moved to ***newdirpath/olddirname***. Note that the two arguments cannot be in the same path. For example:

```
% mvdir x/y x/z
```

is ok, but

```
% mvdir x/y x/y/z
```

is not ok.

6.5.7 Remove a Directory: rmdir

You can remove a directory with the **rmdir** command. The directory must contain no files or subdirectories, and you must have write permission to the parent directory.

```
% rmdir dirname ...
```

You can use an absolute or relative pathname.

You can also use **rm -r** as described in Section 6.3.6. **rm -r** will delete a directory, all subdirectories, and all files. **This command should be used with extreme caution.**

For example, the following command deletes the directory **temp**, all subdirectories of **temp** and all files contained in those directories, prompting before each removal, and confirming removal of write-protected files (**-i**):

```
% rm -ir /usr/jones/temp
```

6.6 File and Directory Permissions

6.6.1 File Access Permissions

The UNIX file system allows you to control read, write, and execute access to your files on the basis of user (owner), group, and other (everyone else).[1] In this section we will consider only the standard UNIX file permissions.

Note that in the AFS file system, file permissions are mediated by Access Control Lists (ACLs) that are set on a directory level. The standard UNIX file permissions don't apply in this case except for the owner permissions, which apply to all users. AFS file permissions are treated in section 7.6.

To determine the current permissions, use the long form of the `ls` command, `ls -l`. Referring to the example below, the nine characters immediately following the first field represent the one-bit flags known as the *mode bits* that control file access. A dash indicates a bit is not set, `r` stands for **r**ead access, `w` for **w**rite access, and `x` for e**x**ecution access. The first set of three characters refer to owner permission, the middle three for group permission, and the last three for all other user classes.

```
    total 251
    drwxr-xr-x 3 nicholls g020c 512   May 2  08:53 Tools
    drwxr-xr-x 2 nicholls g020c 512   May 2  09:01 bin
    -rw-r--r-- 1 nicholls g020c 446   May 4  14:09 defaults
    -rw-r--r-- 1 nicholls g020c 95418 May 1  17:42 intro.lpr
    -rw-r--r-- 1 nicholls g020c 0     May 10 17:51 lsout
    -rw-r--r-- 1 nicholls g020c 6683  May 1  16:46 man.lpr
-rw-r--r-- 1 nicholls g020c 12258 May 9  16:16 out
```

In the example, ignoring the directory files (which have a `d` in position 1), the owner has `rw` access to the files, whereas group and others have read (`r`) access only.

chmod

The `chmod` command, which stands for **ch**ange **mod**e, is used to change access permissions of a file or directory:

`% chmod` ***mode filename ...***

or

`% chmod` ***mode directory ...***

In the *absolute form* of the mode where the level of protection is specified in octal format, ***mode*** looks like 741 or 554, for example, where each of the three octal numbers represents the sum of the permissions granted to its class: user, group, and other, in that order. The three types of permission have the values:

read	4 (100 octal)
write	2 (010 octal)
execute	1 (001 octal)

For example, a mode of 741 means owner can read, write, and execute (4+2+1=7); group can read (4+0+0=4); and others can execute the file (0+0+1=1).

1. Note **o** is for **other** and not for **owner** as on VMS.

To give this permission to a file `test`, you would enter:

```
% chmod 741 test
```

You can use an alternate form of ***mode*** in the **chmod** command in which ***mode*** is a three-character field specifying an action to be taken. The action is to add or subtract one or more permissions from one or more user classes. It takes the form:

who operator permission(s)

These three positions within the field take the following characters:

who	represents the user class or classes; it takes any combination of **u**, **g**, **o**, and **a** for **u**ser (user is really the owner), **g**roup, **o**ther and **a**ll, respectively, where *all* includes the three individual classes
operator	+ or - for adding or subtracting permissions, or = for setting a specific permission and resetting all other permissions for the specified user class(es)
permission(s)	any combination of **r**, **w**, and **x** for **r**ead, **w**rite, and e**x**ecute, indicating the permissions to be permitted, denied, or reset.

Examples of the **chmod** command:

- Remove group execute permission to the file `progs`:

  ```
  % chmod g-x progs
  ```

- For the files `out` and `out1`, add group read and write, and deny write to other:

  ```
  % chmod g+rw,o-w out out1
  ```

- Set group read permission and reset all other group permissions to `myfile`:

  ```
  % chmod g=r myfile
  ```

Note that classes of users or levels of protection not specified in a command are not modified in this form of the command (with the exception that **=** resets other permissions).

umask

With the **umask** command you can specify a mask that the system uses to set access permissions when a file is created. In order to understand **umask** you need to know that access permission at file creation is application-dependent. Each command or application sets a file permission in its *open* command.[1] The system then "subtracts" any user-defined mask, resulting in the final access permission for the file. You can set a umask by this command:

```
% umask [ooo]
```

where ***ooo*** stands for three octal digits. The user-specified "mask", ***ooo***, has the same positional structure as described above for **chmod**, but specifies permissions that should be **removed** (disallowed).

For example, a mask of 022 removes no permissions from owner, and removes write permission from group and others. Thus a file normally created with 777 would become 755 (this would appear as `rwxr-xr-x` in the format put out by the command **ls -l**). The following command could be put in your `.cshrc` or `.profile`.

```
% umask 022
```

The meaning of permissions applied to directories is described in Section 6.6.2.

1. Normally only the loader creates files with execute permission.

6.6.2 Directory Permissions

See section 7.6 for AFS systems.

You can grant or deny permission for directories as well as files, and protection assigned to a directory file takes precedence over the permissions of individual files in the directory.

- Read permission for a directory allows you to read the names of the files contained in that directory with the **ls** command, but not to use them.
- Write permission for a directory allows you to create files in that directory *or* to delete any file in the directory, regardless of the file protection on the files themselves. It does not allow you to see the files or use them without **r** and **x** directory permission. In other words, write permission to a directory allows you to alter the contents of the directory itself, but not to alter, except to remove, files *in* the directory (which is controlled by the file's permissions).
- Execute permission allows you to list the contents of the directory.

File access permissions of directory files are changed with the **chmod** command (see section 6.6.1).

6.7 Temporary Directories

By convention, there are directories named `/tmp` (and sometimes `/usr/tmp`) where programs and users can store temporary files. Many programs (e.g., compilers) write temporary files there or in the area specified by the environment variable *TMPDIR*. Since these are public areas, it is necessary to manage this space, which means that you cannot count on files being retained in these directories.

Many systems on site have fairly small `/tmp` areas and therefore you must be careful not to fill up this space. In general, you should only use `/tmp` for very temporary, small files. On many systems files in `/tmp` will disappear after a reboot or after existing for a week. You can set *TMPDIR* to a different location if there is not enough space in these areas.

Contact the administrators of the particular system to find out what the current policy is on the machine.

Chapter 7: The AFS File System

Fermilab is using the AFS (Andrew File System) as a distributed file service model, and it is installed on several machines on site in a production environment, including the FNALU cluster. This chapter discusses the basic concepts of AFS and provides information on the commands used to manage your files and directories in the AFS environment.

On the Web there is a local collection of information regarding AFS. Look under *The UNIX Operating System* on the *UNIX Resources* Web page. The helpdesk has a small quantity of laminated AFS reference cards that list the syntax of the most common AFS commands. The helpdesk is located in FCC 1W. There are also man pages for AFS commands on the systems running as AFS clients.

7.1 Introduction to AFS

AFS is a modern implementation of distributed file serving. The FNALU cluster is running AFS as a distributed file server in a production environment. Several other systems at Fermilab run AFS as well. AFS has several advantages over older file systems such as NFS (see section 2.6), particularly in the areas of:

- consistency of file organization across the distributed network
- authentication of users (security)
- server management
- volume replication and backup
- server function redundancy

All systems participating in the AFS file system have the same view of the file system. All files for a set of machines known as a *cell* exist under `/afs/{cellname}`. The cellname for Fermilab is `fnal.gov`, thus all files that are part of the AFS file system are located under `/afs/fnal.gov`. There is also a symbolic link (see section 6.3.5) to a shorthand version of the cell name, which is simply `/afs/fnal`. Other cells are also accessible. They are listed as directories under `/afs`, for example the CERN cell at `/afs/cern.ch`.

7.2 How to Determine if AFS is Installed on your System

A special note to CDF and D0 users: Your UNIX systems are *not* configured to use AFS.

To find out if AFS is installed, issue the command:

```
% ps -ef | grep afsd
```

If you get output of the form (note the `/usr/vice/etc/afsd`):

```
    root   305     1  0   Sep 30 ?       14:10 /usr/vice/etc/afsd -stat 2800 -dc
ache 2400 -daemons 5 -volumes 128
    root   306     1  0   Sep 30 ?        5:47 /usr/vice/etc/afsd -stat 2800 -dc
ache 2400 -daemons 5 -volumes 128
```

then it is installed and running on your machine. If you do not get output lines like this, AFS is not installed. However it is still possible that you have access to the AFS file system via a translator[1] service. We discuss translator mode in section 7.9. To check, enter the command:

```
% df |grep afs
```

If no output is returned, AFS is not running on your machine in any capacity. If output is returned and the line begins with a node name preceding `/afs`, for example:

```
fsut01:/afs                          144000000       0   144000000     0%
```

then AFS is running in translator mode. (If the output line begins with `/afs`, then AFS is actually installed on your machine.)

7.3 Issues Related to Login and File Access

7.3.1 Authentication in AFS

On machines running AFS, as on most systems, providing a username and password is sufficient to identify a user as legitimate, and allow the login to succeed. However to access AFS files, you must provide a password recognized by AFS, called a *Kerberos password.* This *authenticates* you to AFS. Once you are authenticated, AFS issues what is known as a *Kerberos token* to your login process. It is having the token that allows you access to the AFS file system. As long as you remain logged on, the Kerberos token "lives" for a period of time set by the AFS administrator of the system; for example in the Fermilab cell it is set to six days.

The token is passed to all subprocesses of the login process (see section 5.1 for an explanation of subprocesses). All normal UNIX interactive operations are therefore automatically authenticated, and access is granted to files in the AFS tree, provided you have the appropriate permissions (AFS permissions are covered in section 6.6.2). The Fermilab standard batch interface **fbatch** provides for token renewal at job execution time, since you can't control when your batch jobs actually run.

Situations occasionally arise in which you are not automatically authenticated (e.g., some remote login methods) or you lose your token (e.g., you remain logged in for more than six days). When this happens and you need to obtain a new token, issue the command string:

```
% pagsh
```

```
% klog
```

pagsh starts a special AFS shell under your login process. **klog** prompts you for your AFS password and obtains a Kerberos token associated with this shell, thus granting authentication and access to files.

A few notes:

1) Running **pagsh** first is much more secure than just running **klog**. It ensures that the token is associated with your pagsh process, and thus with all processes you spawn. **klog**

1. In translator mode, a "translator machine" runs AFS and exports the AFS file tree to other systems.

by itself gets a token associated with your UID, which is not always unique. This could potentially allow another user to share the token, which is undesirable.

2) You cannot enter the commands on one line in the format **`pagsh;klog`**. **`pagsh`** starts a new **sh** shell, and **`klog`** needs to be run at the new shell prompt on the next line.

3) **`pagsh`** changes your shell to **sh**, so you will need to run your preferred shell afterwards (e.g., enter **`tcsh`**, **`bash`**, or **`ksh`** on the command line). You may also then want to source your `.login` and `.cshrc` or your `.profile` and `.shrc` scripts to ensure that your FUE environment is back to normal (see section 4.4 regarding *sourcing* a script, and section 9.4 for information on the login files).

There are Kerberos authentication problems with running programs that spawn jobs external to your login process group. **at** and **cron** fall into this category (they are described in section 5.5.3). You can run the job, but it will not run with authentication, and most likely will not be able to write into `/afs` space. A work-around is available for executing an authenticated **cron** job (send mail to *helpdesk@fnal* requesting the full details of this procedure).

Be aware that being logged on as *root* grants you no special permissions in `/afs` file space; there is no such thing as being "authenticated as root".

7.3.2 Kerberos (AFS) Password

You will have a Kerberos password (sometimes called an AFS password), for any account on a system that uses the AFS file system, for example FNALU. The Kerberos password, which you enter at login time, allows two operations to proceed:

- You can log in to the system.
- Your *Kerberos token* is automatically obtained via the AFS programs **pagsh** and **klog** to allow you access to the file system. The token is attached to your login process.

You can change your Kerberos password using the command:

```
% kpasswd
```

The system will prompt you for the necessary information.

We recommend that you limit your password to eight characters. This enforces consistent behavior in a multi-vendor AFS environment. On some platforms the login program may truncate a long password after eight characters, allowing login to proceed but denying access to the file system.

This command changes your Kerberos password for all systems that run AFS on-site.

7.3.3 Standard UNIX Password on an AFS System

Depending on how the AFS file system was installed, you may or may not have a standard UNIX password in addition to your Kerberos password. In other words, you may *have* a standard UNIX password even if you never need to use it! On FNALU, AFS was installed so that you have only a Kerberos password; no standard UNIX password is defined.

You can find out if you have a standard UNIX password (or an NIS password, see section 2.7) by attempting to change it via the standard UNIX command **`passwd`**. If the command does not succeed (assuming you provide the correct old password if requested), then you do not have a standard UNIX password. Note that the **`passwd`** command returns a different error message on each different UNIX flavor.

If you have both, at login you should provide the Kerberos password so that you obtain your Kerberos token. If instead you provide your standard UNIX password, the system will log you in, but you will not obtain a token and thus will not be able to access AFS files. If the passwords are the same, the Kerberos password automatically takes precedence.

Generally, the Kerberos password is the only password you need. There are exceptions; for example, remote login via a method that doesn't understand Kerberos passwords (e.g., MAC-X to some UNIX platforms). In this case, after logging in using a standard UNIX password, you would need to run the command string **`pagsh`** and **`klog`** as described in section 7.3.1:

7.3.4 Managing your Token

View Active Tokens

To see what tokens you currently hold, you can issue the command:

```
% tokens
```

The output should look similar to this:

```
Tokens held by the Cache Manager:

User's (AFS ID 6302) tokens for afs@fnal.gov [Expires Oct 21 10:22]
User aheavey's tokens for krbtgt.FNAL.GOV@fnal.gov [Expires Oct 21 10:22]
   --End of list--
```

If the output showns no tokens (or only the `krbtgt` token, the second one shown above[1]), then you only have access to (usually a very limited number of) files designated as accessible to the special user *system:anyuser* (a pre-defined AFS protection group; see section 7.7). As its name implies, this designation includes anyone who can access the system (e.g., a user with a standard UNIX password but no Kerberos password).

Get Back an Expired Token

If you remain logged on beyond the set token expiration period, you will find that you no longer have access to AFS files. The system will likely return the message `Permission denied` when you attempt a file operation. To get back the token associated with your login process, issue the commands[2]:

```
% pagsh
```

```
% klog
```

If you are unexpectedly unable to edit your files, try this first! Expired tokens are often the reason for this problem. See the notes in section 7.3.1 regarding these commands.

Destroy a Token

Logging out does not destroy your token; it remains "live" for up to 26 hours afterwards. This is a security risk. Prior to logging out, we advise that you issue the command:

```
% unlog
```

1. The `krbtgt` token was created on FNALU for an application that was never implemented, and it does not affect AFS access. It will probably be removed in the near future.
2. If your token has expired within the previous two hours, you do not need to run **`pagsh`** before **`klog`**.

to destroy the token. If you create a `.logout` file (see section 9.4), you should include this command in it.

Token Issues for Remote Login

One practical issue raised by the Kerberos environment involves the use of the Berkeley networking programs such as **rlogin**, **rsh**, and **rcp**. **telnet** automatically authenticates the user and avoids the issues discussed here. All these utilities are described in Chapter 13.

Normally systems are *equivalenced* to enable the use of the **rlogin**, **rsh**, and **rcp** protocols. The equivalencing of the machines implies that once you are logged into one system, you may log into the equivalenced machines without providing further proof of identity, such as a password. This doesn't fit in with the Kerberos authentication system.

On FNALU, token-passing is available for **rsh** and **rcp**.[1] If you have authenticated into the `/afs/fnal.gov` cell, you can use **rcp** and **rsh** more or less normally between AFS machines; the token will be passed along with the request, and the network communications will be authenticated automatically.[2]

rlogin is more complicated. When equivalencing between two machines is enabled, the **rlogin** protocol does not ask for a password. **rlogin** works, but the remote user is not authenticated at login time, and cannot access most files.

We recommend that you use **telnet** rather than **rlogin** in order to avoid this problem.

If you need to use **rlogin** for some reason, immediately after login issue the commands:

```
% pagsh

% klog
```

as described in section 7.3.1, to obtain authentication.

7.4 AFS File System Commands and man Pages

AFS provides a command (with many options) that allows you to address file system issues such as checking permissions, checking quota, making mount points, finding where a volume is mounted in the file tree, and so on. The AFS file system (**fs**) command is entered in the format:

% fs *main_option -option(s) argument(s)*

Many of the options can be abbreviated, and option flags can often be omitted from the command. Check the man pages, as described below.

To get a list of the main options of the **fs** command, enter:

```
% fs help
```

Here is an edited output listing showing only a few of the options:

```
fs: Commands are:
listacl         list access control list
listquota       list volume quota
lsmount         list mount point
quota           show volume quota usage
```

1. If you are on a non-FNALU node, check with your system administrator before running these utilities.
2. This is set up with the usual UNIX method of using `/etc/hosts.equiv` or `.rhosts` to equivalence systems.

```
rmmount         remove mount point
setacl          set access control list
setquota        set volume quota
whereis         list file's location
whichcell       list file's cell
```

To get usage information on a particular **fs** option, enter:

% fs *option* -help

For example:

% fs setacl -help

```
Usage: fs setacl -dir <directory>+ -acl <access list entries>+ [-clear ] [-negat
ive ] [-id ] [-if ] [-help ]
```

Man pages are available on-line on the AFS systems[1], but you'll find that the names of the manual pages are a little less than clear. Since the **fs** command is *one* command with several main options (e.g., **setacl**, **listacl**), to get the man pages you must attach the command and the main option via an underscore (_):

% man *command_option*

where ***command*** is **fs**, for example:

% man fs_setacl

The underscore is only used with the **man** command, not when issuing the command itself.

7.5 AFS Volumes and Quota

UNIX divides disks into partitions. AFS further divides partitions into subsections called volumes. A volume houses a subtree of related files and directories. Normally, volumes are considerably smaller than traditional file systems. For example, each user's home directory would normally be stored in a separate volume. Large sub-directories are further sub-divided. You do not need to know which file server houses any volume. AFS locates volumes automatically.

To examine the quota on a volume within AFS, the **fs listquota** command may be used. You can request information on several directories at a time. For example the following command requests information on the current working directory (**.**) and on another one specified via an environment variable (see section 9.1):

% fs listquota . $UAFWWW

```
Volume Name             Quota     Used    % Used   Partition
room.aheavey           130000   126024       97%<<       75%    <<WARNING
files.reports.UNIX    2000000    77370         4%         63%
```

The output includes the name of the volume containing the specified directory(ies), the quota size, amount used, percent used, and the percent of space used on the partition containing the volume. You might also get a warning! All sizes are in kilobytes.

1. If you have trouble finding the man pages for AFS commands on your node, check your *$MANPATH* variable to see that it includes `/usr/afsws/man`.

7.6 File and Directory Permissions

7.6.1 File Permissions

File permissions work quite differently from those in standard UNIX, which are described in section 6.6.1. In AFS, you can use the `chmod` command just as you would in a standard UNIX file system. However, it behaves differently. Although in AFS all the permission bits on a file may be examined or changed, **only the owner bits are actually used in AFS, and they apply to all users of the file (as permitted by users' ACL settings; see below).** To turn off write access to a particular file by all users, including the owner, you just need to turn off the owner write bit of the file.

7.6.2 Directory Permissions via Access Control Lists (ACLs)

All other AFS permissions are done with Access Control Lists (ACLs) which take effect at the directory level only. Every directory has its own ACL that defines who can access the directory and its files. Each entry in an ACL consists of a username or an AFS protection group paired with a set of permissions (e.g., read, write). An AFS protection group is simply a list of usernames grouped to share a set of permissions in one or more ACLs. If a user is in two or more ACL entries (e.g., is a member of two groups listed in the ACL) with different permissions assigned, the user gets the union of the permissions.

The permissions granted in a directory's ACL represent the *maximum* permissions. If a file in the directory has more restrictive permissions set, the user is limited by the restrictions on the file. If a file has more lenient permissions set, the user is limited by his ACL entry.

ACL rights include:

`l`	lookup rights (allows user to issue an `ls` command on files in the directory, examine the directory's ACL, and access the directory's subdirectories which are protected by their own ACLs)
`i`	insert rights (allows user to create new files or copy files into the directory)
`d`	delete rights (allows user to remove files or move them to other directories)
`a`	administrator rights (allows user to change the ACL for a directory; note that you always have this right for your home directory even if you accidentally remove this ACL.)
`r`	read rights (allows user to look at the directory's contents and to read the data in the files contained in the directory)
`w`	write rights (allows user to modify the contents of the files in the directory and to change the UNIX mode bits with the command `chmod`)
`k`	lock rights (allows user to run programs that need to *flock* files in this directory; see the man pages for **flock**)

Rights may also be referred to by special names that designate commonly-assigned combinations of rights. These are called *combination rights*. The defined combination rights are:

`write`	all rights but `a` (i.e. `lidrwk`)
`read`	`l` and `r` rights only
`all`	all rights (i.e. `lidarwk`)
`none`	no rights; this removes the group's or user's entry from the ACL entirely

Combination rights can be used in commands, as shown in the examples in section 7.6.2.

A couple of notes:

- In general, users are granted all permissions on their home directories.
- When a child directory is created, it inherits the parent directory's ACL. The child directory's ACL can then be changed. Users of the child directory must have at least lookup rights (`l`) on the parent directory.

Examining a Directory's ACL

You can examine a directory's ACL rights with the command:

`% fs listacl` ***/path/to/directory***

This returns a listing of all the users/groups that have any permissions on the directory, and what the permissions are. The directory path can be absolute (starting from root) or relative to the current working directory. For example, if you run the command:

```
% fs listacl /afs/fnal.gov/files/wwwdocs/cd/webwg/tools
```

The system returns information in the format:

```
Access list for /afs/fnal.gov/files/wwwdocs/cd/webwg/tools is
Normal rights:
  lauram:www_cd_webwg_tools rlidwk
  nicholls:www_cd rlidwk
  hanson:newsmachine rlidwka
  nicholls:wwwdocs rlidwka
  system:administrators rlidwka
  system:anyuser rl
```

The group `lauram:www_cd_webwg_tools` has read, list, insert, delete, write, and lock permissions in this directory (all but administer permissions), i.e. the group has *write* rights. Any member of that group has these permissions in this directory.

Adding/Changing/Deleting a Directory's ACL

You can modify a directory's ACL for a particular AFS group or for an individual using the **`fs setacl`** command. The **`fs setacl`** command only changes the ACL for a single directory, not for a directory tree. The command syntax is:

`% fs setacl -dir` ***/path/to/directory*** **`-acl`** ***group permission(s)***

where ***group*** is either a group or an individual username. When it is a group, it must be entered in the format `{group_owner}:{group_name}`.

We recommend that you generally define ACL entries for groups rather than individuals; it is much easier to maintain. When you need to add or remove permissions for an individual, it is easier to add/remove the user from one or more groups than to track down every directory for which the user appears in the ACL.

The directory path in the command can be absolute (starting from root) or relative to the current working directory. Any pre-existing permissions for the group or individual are invalidated; the specified permissions collectively become the new set of permissions. The permissions apply to all members of the specified group.

For example, to modify the ACL for the current directory (`.`) to include only read and lookup rights for any user (including unauthenticated users), enter:

```
% fs setacl -dir . -acl system:anyuser rl
```

or, using combination rights syntax:

```
% fs setacl -dir . -acl system:anyuser read
```

See the combination rights (e.g., **read**) in section 7.6.2. The group `system:anyuser` is described in section 7.7.

A note for Web page providers: set the permissions for `system:anyuser` to `rl` on directories containing files that you want to make accessible via a Web browser.

To remove all permissions in an ACL for a particular group (or individual), issue the **fs setacl** command with no permissions, e.g.,

```
% fs setacl -dir /path/to/directory -acl group ""
```

or, using combination rights syntax:

```
% fs setacl -dir /path/to/directory -acl group none
```

See **man fs_listacl** and **man fs_setacl** for further information. (Note the underscores in the **man** command.)

7.7 AFS Protection Groups

An AFS protection group is a list of usernames grouped to share a set of permissions on one or more directories. Any user can include any existing protection group in any ACL within your AFS cell. A protection group is designated in the format:[1]

`{group_owner}:{group_name}`

AFS provides three predefined protection groups:

`system:anyuser` This is similar to world permissions in UNIX. Any AFS user (anywhere in the world, and not necessarily authenticated) can access files or directories, according to the permissions granted (e.g., read, write).

`system:authuser` This is a more restrictive version of `system:anyuser`. Only users who have authenticated within the local cell (`/afs/fnal` at Fermilab) may access files, according to the permissions granted (e.g., read, write).

`system:administrators`
This group includes only the few people in the `/afs/fnal` cell authorized to administer AFS.

As determined by your project's `/afs` area manager(s), you may need to manage, and possibly create, protection groups.

Groups can be owned by other groups or by individual userids. Group members often are not allowed to add or remove other members of the group. If a group is owned by a group, then all the members of the *owner group* can by default add or remove other members from the *owned group*. This can avoid problems when key individuals are unavailable. Having one group consisting of a few key individuals, and using this group as the owner for all your other groups is a nice, neat way to organize your groups. Find out from your `/afs` area manager how group ownership and permissions are assigned within your project or on your system.

---

1. You may encounter groups that do not have an owner prefix; these are special groups created by the system administrators.

AFS provides the **pts** command (protection server) for group-related tasks. Like the **fs** command, **pts** has several main options. Also like the **fs** command, you need to use an underscore between **pts** and the main option to access the man page. Issue the command **pts help** to list the main options (this list has been abbreviated to contain only the options we discuss in this section):

```
pts: Commands are:
adduser         add a user to a group
chown           change ownership of a group
creategroup     create a new group
delete          delete a user or group from database
examine         examine an entry
listowned       list groups owned by an entry or zero id gets orphaned groups
membership      list membership of a user or group
removeuser      remove a user from a group
setfields       set fields for an entry
```

7.7.1 Permissions for Performing Group-Related Tasks

Group characteristics (e.g., membership, ownership) can only be seen and/or modified according to the permissions set on the group. We refer you to the man page for **pts_setfields** (notice the underscore) for the full story, and present only a brief explanation here.

Every group has a set of five access flags, which represent permissions for performing sensitive tasks regarding (1) status, (2) ownership, (3) membership, (4) adding members, and (5) removing members. There is a **pts** main option associated with each of these tasks:

status (s)	**pts examine**
owned (o)	**pts listowned**
membership (m)	**pts membership**
add (a)	**pts adduser**
remove (r)	**pts removeuser**

Each flag has one of three possible values: its first letter in lowercase, its first letter in uppercase, or a hyphen. The value determines which users can issue the corresponding command option for the group as follows:

lowercase letter (`s`, `o`, etc.)	all members of the group
uppercase (`S`, `O`, etc.)	all users (i.e. `system:anyuser`)
hyphen (`-`)	group owner and members of `system:administrators` only

As an example, we'll issue a **pts examine** command and examine its output:

```
% pts examine lisa:uss-group
Name: lisa:uss-group, id: -316, owner: lisa, creator: hanson,
membership: 14, flags: S-M--, group quota: 0.
```

The permissions information is contained in the `flags` entry. The flags `S-M--` are the default flags when a group is created (all users can check status and membership information, only group owner and administrators can verify ownership and add/remove group members).

If you can't successfully issue one of the **pts** command options, check the access flags! Of course if you can't issue **pts examine** to check the flags, then you don't have status permissions for the group.

7.7.2 Listing Information about Groups

List Members of a Group

To list the members in a group, use the command:

% pts membership ***group***

For example:

```
% pts membership lauram:www_cd_webwg_tools
```

returns the output:

```
Members of lauram:www_cd_webwg_tools (id: -454) are:
   nicholls
   hathaway
   stolz
   george
   lauram
   dwalsh
   nelly
```

List Groups in which an Individual is a Member

To list the groups to which an individual belongs, again use **pts membership**, but with the user's id as the argument:

% pts membership ***username***

For example:

```
% pts membership aheavey
```

```
Groups aheavey (id: 6302) is a member of:
  nicholls:www_reports
```

List Groups Owned by Group or Individual

To show what groups a particular group or user owns, issue the command:

% pts listowned ***group***

where ***group*** is actually either a group or an individual username. If you try to list groups owned by someone other than yourself, you may find that you do not have permission to do so.

Here are a couple of examples. To check groups owned by the *group* `nicholls:wwwdocs`, issue the command:

```
% pts listowned nicholls:wwwdocs
```

Output is returned in the format:

```
Groups owned by nicholls:wwwdocs (id: -306) are:
  nicholls:www_cd_support
  nicholls:www_cd_mgmt
  nicholls:www_faw_events
  nicholls:www_orgs_folkclub
  nicholls:www_directorate
  nicholls:www_cd_ups
  nicholls:www_cd_webwg
```

To check groups owned by the *individual user* `lauram`, issue the command:

```
% pts listowned lauram
```

Output is returned in the format:

```
Groups owned by lauram (id: 1866) are:
  lauram:wwwmachine
  lauram:expwwwmachine
  lauram:expwwwadm
```

Show Group Ownership

To find a group's owner, use the command:

```
% pts examine -name group
```

This is helpful to determine if a group is owned by an individual or a group. For example, to find the owner of the group `nicholls:www_reports`, run the command:

```
% pts examine nicholls:www_reports
```

```
Name: nicholls:www_reports, id: -378, owner: nicholls:wwwdocs, creator: hanson,
  membership: 5, flags: S-M--, group quota: 0.
```

Its output in the entry `owner` indicates that it is owned by a group (`nicholls:wwwdocs`), not by the individual `nicholls`.

7.7.3 Modifying Group Characteristics

Change the Owner of a Group

Note: It is best to change the owner of the group *before* you run **`fs setacl`** to add directory permissions for the owned group.

You can change ownership of a group using the command:

```
% pts chown -name owned_group -owner owner_group
```

Let's take for example the group `owner1:groupname1`, where `owner1` is an individual. We want to change its ownership to a group. The group we want to own it is designated `owner2:groupname2`. We issue the command:

```
% pts chown -name owner1:groupname1 -owner owner2:groupname2
```

The owned group is now designated `owner2:groupname1`. Notice that it takes its owner designation from the owner group, and maintains its former group name. Here's a more real-life example for clarity:

```
% pts chown -name lauram:www_cd_webwg_tools -owner \
  nicholls:wwwdocs
```

The old `lauram:www_cd_webwg_tools` is now designated `nicholls:www_cd_webwg_tools`.

You can change a group's ownership to itself (and set the group's access flags appropriately if needed) to allow all members of the group to add/remove other members and perform other administrative tasks. To change the group's ownership to itself, issue the **`pts chown`** command with the same group as both arguments:

```
% pts chown -name nicholls:wwwdocs -owner nicholls:wwwdocs
```

The group designation `{group_owner}:{group_name}` does not change. If you need to reset the group's access flags, see **man pts_setfields**.

Note that there is a potentially confusing consequence of the way the group names change. All groups *look like* they're owned by individuals. You can always issue the command:

```
% pts examine -name group
```

to determine if the owner is an individual or a group, as shown under **Show Group Ownership** in section 7.7.2.

Add a Member

To add a member, use the command:

```
% pts adduser -user username -group group
```

For example:

```
% pts adduser -user nelly -group lauram:www_cd_webwg_tools
```

The new member (`nelly`) must have an account on the system/cluster that mounts the AFS files he or she needs to access.

Remove a Member

To remove a member from a group, use the command:

```
% pts removeuser -user username -group group
```

Create a Group

Check with your `/afs` area manager before creating new groups. As groups proliferate, system management can become more difficult.

To create a new AFS protection group, use the command:

```
% pts creategroup -name group
```

or, leaving off the **-name** option flag for simplicity:

```
% pts creategroup group
```

Always enter a group in the format `{group_owner}:{group_name}`; don't enter only the `{group_owner}` portion. By default, the group owner is yourself.[1]

As an example, user *lauram* could run the command:

```
% pts creategroup lauram:www_cd_webwg
```

Remove a Group

To remove a group, use the command:

```
% pts delete -nameorid group
```

---

1. There is an option (**-owner**) to set the owner to another individual or a group, but we recommend that you just use **chown** afterwards as described in section 7.7.3.

For example:

```
% pts delete -nameorid lauram:www_cd_webwg_tools
```

7.8 Implications of ACLs and Examples

The implementation of security in our Fermilab AFS cell is based on the notion that sharing information is more important than trying to protect it. Therefore, in most cases, the default has been to set ACLs to have the least security that is still reasonable. As currently implemented, all user home directories come with their ACL set so that `system:anyuser` has **`rl`** (read and lookup) permissions. A `Mail` subdirectory (used by the **MH** mail readers) is provided with more secure permissions.

The practical implication of this is that *anyone* on the internet running an AFS client can read your files, unless you change the ACL. Home directories are *writable* only by their owners (that is, the owner has **`rldiwka`** permission), but the world can read them. This is probably fine in many cases, but you should be aware of it and protect your files as you see fit, according to the guidelines presented below.

7.8.1 Protecting your Subdirectories

You can protect any single directory by changing its ACL to turn off permission for `system:anyuser` as well as for other users or groups that should be denied permissions. For example, if you use the mail reader **pine**, you may want to protect the message subdirectory `mail`. To make it completely inaccessible by `system:anyuser`, you'd issue this command:

```
% fs setacl $HOME/mail system:anyuser none
```

On the other hand, if you need to allow others to write into any of your directories, the default permission is too constraining. Say you are in a collaborative effort with user *mrchips*. You could allow him full permission in your `shared` directory by issuing the command:

```
% fs setacl $HOME/shared mrchips all
```

Recall from section 7.6.2 that if a user is in two or more groups that have different permissions on a directory, the user gets the union of the permissions.

Also, recall from section 7.6.2 that the **`fs setacl`** command only changes the permission for a single directory. If you have a directory hierarchy on which you want to change permissions, you'll have to use a UNIX command that traverses down the tree and changes all the directories as it goes. The **`find`** command can be used (see section 6.4.1), but it must be used judiciously in the AFS environment! **This is not recommended for inexperienced UNIX users (see section 7.11.2).** As an extension of the above example, say you had a directory hierarchy under `shared` to which you wanted to allow *mrchips* full access. The **`find`** command could be used instead of **`fs setacl`**, as follows:

```
% find $HOME/shared -type d -print -exec fs setacl -dir {} -acl\
  mrchips all \;
```

This would traverse down from the `shared` directory, changing the ACL for each of the directories it finds. The **`-print`** argument causes the system to print out all the directories the command encounters, allowing you to monitor the progress.

7.8.2 Protecting your Home Directory

> We strongly recommend that you make your home directory world readable, and simply keep your private files in protected subdirectories. That said, ...

We do *not* recommend that you set the ACL on your home directory such that `system:anyuser` has no permissions (i.e. combination rights **`none`**) in order to keep your top level directory private. There are at least a couple of undesirable consequences:

- If you ever managed to log in unauthenticated, you wouldn't be able to enter your home directory. In fact you might not be able to log in at all, depending on the platform.
- The dot files in your home directory (e.g., `.login`, `.profile`) would be unreadable by unauthenticated system processes, which could cause them to break. For example, if `system:anyuser` does not have at least **`rl`** permissions on your home directory, the **sendmail** program will not be able to read your `.forward` file, and your mail forwarding will break.

If for some reason you really want to protect your home directory, you can do so to the extent that only **`l`** (lookup) permission is granted. However, you must make sure that any files that *must* be world readable, such as your `.forward` file, remain accessible. **Be aware that it is not always obvious which files must remain world readable in order to preserve the behavior of your environment.** You can protect your home directory as follows (Proceed with caution!):

1) Every AFS home directory is created with a subdirectory called `public`. Move the files that must remain world readable into this directory.

2) For each file moved into `public`, create a symbolic link in your home directory to the file in the `public` directory. Use the same filename.

3) *After* all the necessary files are moved and linked, *then* shut off all permissions except **`l`** (lookup) on your home directory.

Note that you *must* leave the **`l`** permission turned on or programs won't be able to find the file in `public` via the symbolic link.

Here is a sample session, assuming the only file that must remain world readable is `.forward` (there would actually be many files). It would be run from the user's home directory:

```
% mv .forward public/.forward

% ln -s public/.forward .forward

% fs setacl . system:anyuser l
```

7.9 AFS in Translator Mode

If your machine is accessing AFS via a translator node, you do not get authenticated when you log in, and in fact you can't run the **pagsh** or **klog** programs discussed in section 7.3.1. You cannot access your AFS login area. You only have access to directories for which an ACL is defined for `system:anyuser`. You have access to files in those directories according to the ACL entry for `system:anyuser` and the file owner bits, as usual.

At Fermilab most of the **UPS** products are set with *read* permissions (**`rl`**) for `system:anyuser`, thus allowing access to products maintained in the `/afs` products area from a machine running in translator mode. This is not true for products that are site-licensed (e.g., **edt**), which are made accessible only to users on site.

7.10 File Locking in AFS

The file locking mechanism in AFS does not really follow POSIX semantics. There are a few issues to mention:

- Files may only be locked as a whole; regions of a file may not be locked.
- File locking only works properly and reliably from a single system. If a file is locked from one client and an attempt is made to access the file from another client, the error `EWOULDBLOCK` is returned.
- There is no deadlock prevention in AFS, so deadlock situations can occur with file locking.
- Any program that attempts to use byte-range file locking in AFS will get a message from the cache manager warning that other users may be accessing the same file. Usually these messages can be safely ignored.

Generally we don't recommend including applications that depend on file locking in the AFS file space. Contact the helpdesk for more information or for resolution of a problem.

7.11 Frequently Asked Questions

7.11.1 Lost Access to Files

Why can't I access files I'm supposed to be able to edit?

First see what permissions you have by checking:

- which groups have `rlwidk` permissions on the directory
- that you are in at least one of these groups
- that the owner of the file has the standard **`rw`** UNIX permissions (see section 6.6.1)

Remember that authentication lasts only a set period of time (6 days in `/afs/fnal.gov`). If your authentication has expired, you will not have access to your files. You can reauthenticate by running **`pagsh`** and **`klog`** (see notes in section 7.3.1). Also we encourage you to use **telnet** in (which always authenticates to AFS) instead of using **rlogin**. See section 7.3.4 and chapter 13 for information on these utilities.

7.11.2 AFS and the UNIX Command "find"

Why shouldn't I use "find" in AFS space?

You should be *very careful* about using any command that traverses the file system on a machine that has `/afs` mounted. Be aware that a **`find`** on your system starting at root (**`/`**) will traverse the *whole* AFS file tree, including all the other AFS sites mounted on our cell. This is particularly problematic on some workstations, like Solaris 1 Suns, which by default run a nightly **cron** job that traverses the whole file system. Also note that the **`-mount`** and **`-xdev`** options (e.g., **`find / -mount ... -print`**) won't recognize an `/afs` file system boundary; **`find`** can't tell the difference between local files and AFS files. The **`find`** command is discussed in section 6.4.1.

7.11.3 Error Messages

What does it mean if I get an error message like this:

```
afs: Waiting for busy volume 536870945 in cell fnal.gov
```

This is an error message from AFS that indicates that you are trying to access a volume that is busy. There may be a number of perfectly normal reasons for this. It probably means either that your volume is in the process of being cloned for a nightly backup, or that one of the system administrators is in the process of moving your volume to a different disk because the one you are on is filling up. Normally the process that generated the error will just hang harmlessly for a few minutes until whatever locked the volume is finished. If this goes on for more than 20 minutes or so, contact the helpdesk and inquire about what is going on.

7.11.4 Retrieving Old Files

What if I need to retrieve yesterday's copy of a file?

Daily backups of the entire Fermilab cell are available from `/afs/fnal.gov/files/backup/`. For example if your home area is `/afs/fnal.gov/files/home/room3/joe`, you should be able to find yesterday's files in `/afs/fnal.gov/files/backup/home/room3/joe`. The backups are done at 12:45 a.m. seven days a week.

Not all of the volumes are readily available (as of November 1997), but work on that is proceeding. In the interim, if you cannot locate your files, contact the helpdesk to request that your backup volume be mounted so that you can access it.

7.11.5 Link Failure

Why did my link fail?

Hard links can be used only within an AFS volume, not across volumes. Generally, you should use symbolic links. Links are discussed in section 6.3.5.

Chapter 8: Printing

This chapter covers the standard FUE print utility **flpr**, as well as filter programs and techniques available for formatting the output prior to printing. **flpr** is the Fermi implementation of the standard UNIX **lpr** utility. Most software applications supplied by the Computing Division use **flpr** as a default.

Note that many X-windows applications allow you to specify in a pop-up window the print command you want to use.

8.1 The FUE Print Command: flpr

The Fermilab print utility **flpr** (pronounced "flipper") implements the **lpd** printing protocol, and can be used to print UNIX files to any print server using this protocol, including VMS print queues and most UNIX systems. Related commands are `flpq` (to check the print queue) and `flpk` (to kill a submitted print job; not supported on many hosts). The format of each command is:

```
% flpr [options] [file] ...

% flpq [options] [users|jobs] ...

% flpk [options] [users|jobs] ...
```

The options are described in the man pages. All three programs will display a list of their options when given an argument of `-\?`. Which options actually function depends on the functions supported by the **lpd** host.

The following command produces the version of **flpr** in use, when and where it was created, the **flpr** defaults in effect, and the command format:

```
% flpr -\? -v
```

To print a file, generally you will just need to enter the following:

```
% flpr -q queue file
```

This assumes that your implementation of **flpr** uses FNPRT.FNAL.GOV as the default host computer (most do), and that your printer is known to FNPRT (most are). If the `-q` option is not specified, the queue is taken from (in order of precedence):

1) environment variable *FLPQ*

2) user control file (`.flprrc`)

3) system control file(s)

4) data compiled into **flpr**

A full list of queues supported by FNPRT is available by running the command **obtain printer** to create the file `flp.printers` in your directory[1].

If your printer host is not known to **flpr**, you can use the option **-h** to specify it on the command line, set your own default printer host with the environment variable *FLPHOST*, or put an appropriate entry in your `.flprrc` user control file.

You can set environment variables such as *FLPQUE* and *FLPHOST* with the **setenv** command for the C shell family (see section 9.1):

% setenv FLPQUE *queue*

% setenv FLPHOST *host*

and for the Bourne shell family:

$ FLPQUE=*queue*; export FLPQUE

$ FLPHOST=*host*; export FLPHOST

A useful feature of **flpr** is the capacity to use *nicknames*. A nickname specifies the printer host computer and the queue as a pair. If you often use a printer unknown to FNPRT, this feature may be handy for you. You would then enter a print command in this format:

% flpr -P *nickname file*

The `flp.printers` file mentioned above defines many nicknames, most or all of which are associated with the host FNPRT, and is therefore more useful as a queue reference for the **-q** option. An example nickname is `wh10w_lw`, which is defined as:

```
printer wh10w_lw fnprt.fnal.gov wh10w_lw
```

Why are printer nicknames useful? If you're using a printer not recognized by the **flpr** defaults, you'll need to specify both host and queue unless you use a nickname. Also, we recommend that you use nicknames rather than queue names in shell scripts. If a printer fails or is removed, then its associated nickname can be redefined, and the scripts don't need to be changed. If a queue name is specified in a shell script, then this capacity does not exist. You can define a nickname yourself, or if a widely-used printer is down, it may be appropriate for your system administrator to establish a substitute printer by changing the nickname in `flp.printers`.

A printer nickname does not have to be the same as the printer queue name. It may be a short, easily remembered name, such as *ps* for a postscript printer or *lp* for a text printer. Such personal printer nicknames can be established in your personal **flpr** control file, `$HOME/.flprrc`.

As an example `.flprrc`, assume you use the FNPRT queues `b0trwqms_hp` for text printing, and `b0trwqms_ps` for Adobe PostPscript printing. You want to use the nicknames *lp* for text printing and *ps* for PostScript. Further, you would like the print jobname to contain identifiers for your login name and the file being printed. You could build a `.flprrc` as follows[2]:

```
host fnprt.fnal.gov
queue b0trwqms_ps
identifier %l$%b
printer ps fnprt.fnal.gov b0trwqms_ps
printer lp fnprt.fnal.gov b0trwqms_hp
```

1. This file actually specifies printer-queue combinations. These names are the same as the queues when FNPRT is the host, the case for most if not all of the entries.
2. The *identifier* in the file is equivalent to the **-I** option described in the **flpr** man page.

If this file exists, it will be used automatically. **flpr** also accesses any system-wide defaults files, such as `flp.defaults` and `flp.printers` stored in a system directory. The `flp.defaults` file contains default values for the host, username, printer queue, protocol temporary filename format and one or more associations between a nickname and a host and printer queue pair.

Another option for the `flpr` command that may occasionally be useful is `-l` ***login-name***. This specifies the remote login name to be used in case you want another user name associated with the print job. The default name is the same as that on the current system.

From the *UNIX Resources* page on the Web, see *Printing from UNIX at Fermilab* under **Our Local Environment** for information on the Fermilab print server, and how to list and check print queues.

8.2 Pre-Printing Options

a2ps and **psnup**, described below, are now in the **UPS** product **psutils**. You may need to run `setup psutils` before using them. They can be used from the UNIX command line as well as from many applications. **pr** is also available to format the pages before printing with **flpr**. **pr** is explained in the man pages.

a2ps and **psnup** are described in detail in the man pages and in the Fermilab DCD Release Note 41.0 *Users Guide to UNIX Printing Utilities*.

8.2.1 Convert ASCII to PostScript: a2ps

The **a2ps** utility converts ASCII to two-column PostScript by default, and encloses the text in boxes with headers indicating date, time, filename and page number. You also get a line saying "Printed by ***username*** from ***node***". You can override the defaults; see the man pages.

a2ps is useful if the printer can only accept PostScript. You can use it to produce different output formats (e.g., "2-up", where two pages of your file are shrunk to fit on a single sheet). It provides options to add items like line numbers, user, file, and system information to the output. The command syntax is:

`% a2ps [`***global options***`] `***files*** `[`***positional options***`] `***files***

where ***global options*** apply to all files being printed, and ***positional options*** are applied only to the files found in the remainder of the command line (with the exception of `-H` as noted in the man page which only applies to the *next* file).

Here is an example where we pipe the output of `a2ps` to `flpr` (`-p` indicates portrait mode):

`% a2ps -p `***file1*** `| flpr -q wh10w_lw`

8.2.2 Print Multiple Pages per Sheet: psnup

The **psnup** utility takes a PostScript file and prints it "***n***-up". This refers to how many pages get printed on a single sheet. Portrait and landscape mode alternate as ***n*** changes. Occasionally this procedure doesn't work due to problems with PostScript variance.[1] The syntax of the command is:

1. **psnup** inserts PostScript code in front of the PostScript it is given. Because PostScript is a programming language, this doesn't always work. The file may be printed in its original format, it may never be printed, or it may print fine!

% psnup [-p*n*] [-r] [-R] [-s*n*] *files*

where:

-p*n*	the number of pages of PostScript (or *spots*) that should be printed per sheet of paper. ***n*** is constrained to be 2, 4, 8, 16, 32, or 64.
-r	sets the first spot in lower right and progresses horizontally to the upper left. This is handy for when the pages have already been reversed by another program, and you are printing on a printer that reverses pages.
-R	sets the first spot in the upper left hand corner and progresses horizontally to the lower right. This is for non-reversing printers.
-s*n*	***n*** is the number of the spot you want the first page of output to be placed. *The first spot on the page is 0, not 1.*

8.2.3 Set Duplex Mode

Duplex mode refers to printing on both sides of the paper. This is often desirable for larger documents, although not all printers support this feature.

PostScript Files

To print a PostScript file in duplex mode on a printer that supports this feature, you must prepend some specific text to the file. An easy way to do this is to maintain this text in a separate file, and concatenate it to your PostScript file when you are ready to print it in duplex mode. The two-line text file, which we'll call `duplexps` here, must have the following contents:

```
%!
statusdict begin true setduplexmode end
```

To print your PostScript file in duplex mode, enter the command:

% cat duplexps *postscript_file* | flpr -q*duplex_printer*

Text Files

For text files, you need to prepend a different sequence to the file. The technique we present here works with any PCL-based printer with a duplex device in it (assuming no filters are in the way) . Since text files are easy to edit, just add the following line to the top of your file[1]:

```
<ESC>&l1S<ESC>&a0G
```

The trick is to get the *escape* character in your file! The sequence <Ctrl-[> will work for escape, but it must be preceded by a "quoting" character, which differs from editor to editor.

1. After the first ampersand (&), the characters are: lower case L (l), one (1), uppercase S (S). The character before the final G is a zero (0).

For this (these) editor(s):	... enter:
vi	<Ctrl-v> <Ctrl-[>
emacs	<Ctrl-q> <Ctrl-[>
nu/TPU, fermitpu	<Ctrl-v> <Ctrl-v> <Ctrl-[> <Ctrl-[>
EDT+	<Ctrl-[> <Ctrl-[>
NEdit	Use Insert Control Character from the Edit menu; escape is 27 decimal. The escape character may not echo on your screen depending on your font.

... to insert an escape character. Then print the file in the usual manner to an appropriate printer.

8.3 Other Print Utilities

There are two other traditional printing systems under UNIX. One of these is **lpr**, upon which **flpr** is based. **lpr** requires that each printer be defined in the local `/etc/printcap` file, which must be done by the system administrator. One of **flpr**'s advantages is that no such definition is required since the host name and print queue can be specified on the command line. **lpr** is the default print command used by some applications, however. You can obtain in your current directory the file `printcap` which contains all printcap entries for print queues defined on FNPRT by executing the command:

```
% obtain printcap
```

The other traditional printing system is called **lp**. This has its own set of files which must be set up by system administrators and is recommended for use only in situations where the other options are not available to you.

Chapter 9: Working Environment

This chapter describes the methods used to set up your working environment in UNIX. Some of these are standard UNIX (e.g., shell and environment variables), and some are provided and/or customized by FUE (e.g., the login scripts).

9.1 Shell Variables and Environment Variables

Every UNIX process runs in a specific *environment*. An environment consists of a table of *environment variables*, each with an assigned value. When you log in certain *login files* are executed. They initialize the table holding the environment variables for the process. (Exactly which files run will be made clear later in this chapter.) When this file passes the process to the shell, the table becomes accessible to the shell. When a (parent) process starts up a child process, the child process is given a copy of the parent process' table. Environment variable names are generally given in upper case.

The shell maintains a set of internal variables known as *shell variables*. These variables cause the shell to work in a particular way. Shell variables are local to the shell in which they are defined; they are not available to the parent or child shells. Shell variable names are generally given in lower case in the C shell family and upper case in the Bourne shell family.

9.1.1 C Shell Family

The C shell family explicitly distinguishes between *shell variables* and *environment variables*.

Shell Variables

A shell variable is defined by the **set** command and deleted by the **unset** command. The main purpose of your `.cshrc` file (discussed later in this chapter) is to define such variables for each process. To define a new variable or change the value of one that is already defined, enter:

% set *name=value*

where ***name*** is the variable name, and ***value*** is a character string that is the value of the variable. If ***value*** is a list of text strings, use parentheses around the list when defining the variable, e.g.,

% set *name=(value1 value2 value3)*

The **set** command issued without arguments will display all your shell variables. You cannot check the value of a particular variable by using **set *name***, omitting ***=value*** in the command; this will effectively unset the variable.

To delete, or unset, a shell variable, enter:

% unset *name*

To use a shell variable in a command, preface it with a dollar sign (**$**), for example ***$name***. This tells the command interpreter that you want the variable's value, not its name, to be used. You can also use **${*name*}**, which avoids confusion when concatenated with text.

To see the value of a single variable, use the **echo** command:

% echo *$name*

If the value is a list, to see the value of the ***n***th string in the list enter:

% echo *$name*[*n*]

The square brackets are required, and there is no space between the name and the opening bracket.

To prepend or append a value to an existing shell variable, use the following syntax:

% set *name=prepend_value*${*name*}

or

% set *name*=${*name*}*append_value*

Note that when a shell is started up, four important shell variables are automatically initialized to contain the same values as the corresponding environment variables. These are *user*, *term*, *home* and *path*. If any of these are changed, the corresponding environment variables will also be changed.

Environment Variables

Environment variables are set by the **setenv** command, and displayed by the **printenv** or **env** commands, or by the **echo** command as individual shell variables are. Some environment variables are set by default (e.g., *HOME*, *PATH*).

The formats of the commands are (note the difference between **set** and **setenv**):

% setenv [*NAME value*]

% unsetenv *NAME*

where ***value*** is interpreted as a character string. If the string includes blanks (i.e. if it encompasses multiple values), enclose the string in double quotes (**"**), e.g.,

% setenv *NAME "value1 value2 ..."*

The current environment variable settings can be displayed using the **setenv** command with no arguments.

To use an environment variable in a command, preface it with a dollar sign (**$**), for example ***$NAME***. This tells the command interpreter that you want the variable's value, not its name, to be used. You can also use **${*NAME*}**, which avoids confusion when concatenated with text.

To prepend or append a value to an existing environment variable, use the following syntax:

% setenv *NAME "prepend_value*${*NAME*}"

or

% setenv *NAME "*${*NAME*}*append_value*"

If the pre- or appended value is the value of a preexisting environment variable, enclose the variable name in braces, too, e.g.,

% setenv *NAME "*${*NAME*}${XYZ_VAR}"

Appending and prepending is commonly used with the *PATH* variable, and a colon is used as a separator, e.g.,

```
% setenv PATH "${PATH}:${XYZ_DIR}"
```

9.1.2 Bourne Shell Family

The Bourne shell family does not really distinguish between shell and environment variables. When a shell starts up, it reads the information in the table of environment variables, defines itself a shell variable for each one, using the same name (also uppercase by convention), and copies the values. From that point on, the shell only refers to its shell variables. If a change is made to a shell variable, it must be explicitly "exported" to the corresponding environment variable in order for any forked subprocesses to see the change. Recall that shell variables are local to the shell in which they were defined.

Shell variables are defined by assignment statements and are unset by the **unset** command. The format of the assignment statement is:

$ ***NAME=value*** **[; export** ***NAME*****]** [1]

where there are no spaces around the equal sign (=). The **unset** command format is:

$ unset ***NAME***

where ***NAME*** is the variable name, and ***value*** is a character string that is the value of the variable. If the string includes blanks (i.e. if it encompasses multiple values), enclose the string in double quotes, e.g.,

$ ***NAME="value1 value2 ..."***

The values of all the current variables may be displayed with the **set** command.

To use a variable in a command, preface it with a dollar sign (**$**). This tells the command interpreter that you want the variable's value, not its name, to be used. For example, to see the value of a single variable, enter:

$ echo ***$NAME***

You can also use **${***NAME***}**, which avoids confusion when concatenated with text.

To prepend or append a value to an existing environment variable, use the following syntax:

$** ***NAME=prepend_value$NAME*

or

$** ***NAME=$NAMEappend_value*

Appending and prepending is commonly used with the *PATH* variable, and a colon is used as a separator, e.g.,

```
$ PATH=${PATH}:${XYZ_DIR}
```

---

1. In most cases you will want to include the optional part of the command, so that it reads: ***NAME=value; export NAME***

9.2 Some Important Variables

These variables are important for all shells, unless noted otherwise.

DISPLAY

In order to use an X windows application, the environment variable *$DISPLAY* must be set correctly. Normally the Fermi login files set it correctly for you. Its value is of the form ***node:screen.server***. At Fermilab, this will generally look like ***node*`.fnal.gov:0.0`** where *node* is your machine name.

Finding the value of *name* is different on different types of terminals and workstations:

UNIX workstation	On a local window, run **`funame -n`** to get the name.
NCDX terminal	("X-terminal") Use the TCP/IP Name found under Show Version from the startup menu.
Tektronix X terminal	Enter setup by pressing F3 (the setup key). Select setup. Under Configuration Summaries, select TCP/IP. A screen appears with the nodename. This is the name you want.
VMS workstation	If the workstation is connected directly to the monitor, then use the name of the workstation.

HOME

Your home directory is the top of your personal branch in the file system, and is usually designated by your username, i.e. `/{path}/{username}`. The value of the variable *HOME* is the pathname of your home directory. The command **`cd`** without arguments always returns you to *$HOME*. In all shells except **sh**, the tilde (**~**) symbol used in filename expansion, expands to the value of this variable. For example `~/myfile` is equivalent to `$HOME/myfile`. `~{username}` is equivalent to the $HOME directory of user `{username}`.

PATH

The *PATH* variable lists the set of directories that the shell looks in to find the commands that you enter on the command line. (For the C shell family, the shell variable *path* takes its value from *PATH*.) If the path is set incorrectly, some commands may not be found. If you enter a command with a relative or absolute pathname, the shell will only search that pathname for it, and not refer to *PATH*.

If you include the current working directory, "dot" (**`.`**), in your *PATH*, the shell will always find your current working directory. This allows you to run executable files from your current working directory by typing in only the filename. The Fermi login files include the dot at the end of the path for you.

For the C shell family, see the following line in the `setpath.csh` file (see section C.1.3):

```
set path = ($path .)
```

For the Bourne shell family, see the following line in the `setpath.sh` file (see section C.2.3):

```
PATH = ${PATH}.:
```

See section 9.4 for information on these files. If "dot" is not in your *PATH*, then in order to execute a file, you need to precede the executable filename by **`./`** on the command line. This provides the current directory pathname explicitly.

LINES and COLUMNS

These variables control the number of lines and columns are displayed on your screen. The **csh** family syntax is:

```
% setenv LINES n
```

```
% setenv COLUMNS n
```

to set the number of lines or columns to ***n***.

Note for Solaris (SunOS 4.x) and OSF1: Use instead:

```
% stty -rows n
```

```
% stty -cols n
```

MANPATH

The *MANPATH* variable lists the set of directories that the shell looks in to find man pages.

SHELL

This variable is set to your default shell. Your default shell is determined by the last field in your password entry (see section 4.1.2).

ignoreeof

This shell variable is in **csh**, **tcsh**, **ksh** and **bash**. **sh** doesn't have it. When the *ignoreeof* variable is set, you cannot exit from the shell using <Ctrl-d>, so you cannot accidentally log out. You must use `exit` or `logout` to leave a shell (see section 2.2).

noclobber

This shell variable is in **csh**, **tcsh**, **ksh** and **bash**. **sh** doesn't have it. With the *noclobber* variable set, you are prevented from accidentally overwriting a file when you redirect output. It also prevents you from creating a file when you attempt to append output to a nonexistent file.

noclobber has no effect on utilities such as **cp** and **mv**. It is only useful for redirection. See sections 5.4.2 and 6.3.

9.3 The Alias Command

The `alias` command allows you to create your own names or abbreviations for commands by performing string substitution on the command line according to your specifications. Aliases are recognized only by the shell that invokes them; spawned processes do not "inherit" them.

Never use the actual command syntax as an alias for itself. If for some reason an error occurs and the login file which defines your aliases doesn't run, UNIX executes the standard version of the command. Normally you'd see an error message in this case, but what if you miss it? This can be disastrous. For example, if you are accustomed to using `rm` (*remove file(s)*, see section 6.3.6) as an alias for `rm -i` (*remove file(s), but prompt for confirmation*), when you run `rm` you will expect a confirmation prompt. If the alias didn't get defined you won't get a prompt, and you may end up removing files you need. That is why we suggest `rmi` as an alias for this command.

9.3.1 C Shell Family

The format of the **alias** command is:

% alias [*new* [*old*]]

When you enter ***new*** the shell substitutes ***old***.

The first example causes **ls -l** to be executed when the command **ll** is entered:

```
% alias ll ls -l
```

The next example creates the command **dir** to list directory files only:

```
% alias dir 'ls -l | grep ^d'
```

grep in this case searches for a **d** in the first column of each line.

9.3.2 Bourne Shell Family

Alias

The **alias** command is supported by **ksh** and **bash**, but not **sh**. For the entire Bourne shell family you can use *shell functions* instead of aliases; we discuss these below. The format of the **alias** command is:

% alias *name* = '*alias_contents*'

The first example causes **ls -l** to be executed when the command **ll** is entered:

```
% alias ll='ls -l'
```

The next example creates the command **dir** to list directory files only:

```
% alias dir='ls -l | grep ^d'
```

grep in this case searches for a **d** in the first column of each line.

Shell Functions

The Bourne and Korn shells support shell functions, which are similar to shell scripts in that they store a series of commands for execution at a later time. Shell functions are more quickly accessed than scripts because they are stored in memory instead of a file, and the shell preprocesses them. They can be used in place of aliases in order to be completely portable between **sh**, **ksh**, and **bash**. For information on the format of shell functions and how to use them, refer to the **sh** man pages or a UNIX text.

9.4 Tailoring Your Environment

This section discusses the FUE-customized login files (also called the *Fermi files*) used to set up your UNIX environment. All the files discussed in this section are printed in Appendix C. You will automatically have your own copy of these files in your home directory[1]. The default files exist in `/usr/local/etc` and you can recopy them to your home directory if you ever need to. Once you understand the functions of the various files, you can tailor them to suit your tastes.

Many of these files include sample code that you may want to activate. A pound sign (#) in the first column indicates a comment line. To activate a command line that's been "commented out", remove the `#`.

9.4.1 C Shell Family Fermi Files

The C shell executes *hidden* FUE-customized files at various times in your session. They include the files `.cshrc` and `.login`, which you may choose to further modify.

When you log out, the shell looks for a logout script in your home directory called `.logout`. FUE does not provide this file, but you can create it yourself, and it will get run automatically. This file is not required.

As an example, including the **`clear`** command in your `.logout` file contents clears the screen when you logout.

If your system is running AFS, we recommend including the **`unlog`** command in the `.logout` file. This is explained in section 7.3.4.

.cshrc and fermi.cshrc

Upon logging in, the first file to execute is the `.cshrc` located in your home directory. The shell also executes this file each time you invoke a new C shell, for example when you execute a C shell script or otherwise fork a new process.

Your `.cshrc` file first executes `/usr/local/etc/fermi.cshrc`, which:

- sets up the machine id, type, and operating system
- sets up **UPS**
- establishes a reasonable default *path* (and therefore *PATH*) by running `/usr/local/etc/setpath.csh` (see section C.1.3), and *MANPATH*
- sets *fermimail* as the standard mail alias

`fermi.cshrc` also calls `/usr/local/etc/local.cshrc` which may set other environment variables.[2] You do not have a copy of `fermi.cshrc` in your home directory; it is not designed to require individual customization. See section C.1.2 for the file listing.

The file `.cshrc` should contain all your aliases so that child processes have access to them; many suggested aliases are provided for you to activate, and you can define your own. You can also set shell variables (*noclobber* and *ignoreeof* are already set for you) and parameters that are local to a shell.

---

1. Exceptions are the `fermi.*` and `setup.*` files which are called directly from `/usr/local/etc`.
2. This is a file for things that the local system manager wants to add to the login scripts. It may or may not have been created on your system.

Don't set any environment variables here. Any changes to their values will remain after you terminate a forked process, thus changing your standard environment for the duration of your login session. See section C.1.1 for the default file listing.

.login and fermi.login

The `.login` file is executed only at login time. After execution of `.cshrc`, the `.login` file located in your home directory is run. The default `.login` file first executes the file `/usr/local/etc/fermi.login`.

`fermi.login` performs several actions:

- sets **umask** (default file access permissions) so others can read and execute but not modify or delete your files
- determines the terminal type (and makes "best effort" at determining *DISPLAY* variable)
- sets common terminal characteristics
- sets a host of environment variables

You do not have a copy of `fermi.login` in your home directory; it is not designed to require individual customization. See section C.1.6 for the file listing.

Next, the `.login` file sets your prompt, and sets the variables *history* and *savehist*. You can edit your `.login` to modify your path and/or terminal settings, change the default values of environment variables or create your own, and/or include commands that you want to execute once, at the beginning of each session (for instance **`setup`** ***product*** commands). See section C.1.5 for the default file listing.

.logout

The C shell executes the `.logout` file in your home directory (if you have created one) when you log off the system.

Execute files to modify current session

If you modify your `.cshrc` or `.login` files and you want them to take effect in the current session, you must execute them with the **`source`** command:

```
% source .cshrc
% source .login
```

This is explained in section 4.4.

9.4.2 Bourne Shell Family Fermi Files

The Bourne shell executes *hidden* FUE-customized files at various times in your session. When you log on in the Fermi environment, the `.profile` and `.shrc` files in your home directory are executed for **sh**, **bash**, and **ksh**. Your `.shrc` file is also executed at any time a new **bash** or **ksh** is invoked.[1]

---

1. On some of the more recent OS releases (e.g., AIX+4 and IRIX+6.4, and likely others in the near future) `/bin/sh` is a link (links are described in section 6.3.5) to the korn shell (**ksh**). **ksh** is a superset of **sh**, so this shouldn't present any problems for you. One difference is that your `.shrc` file gets sourced when you run `/bin/sh` scripts.

The name of the file `.shrc` is determined by the *ENV* environment variable which is set to `~/.shrc` in the standard `.profile`, it is not a standard UNIX feature.

.profile and fermi.profile

The `.profile` file first executes `/usr/local/bin/fermi.profile`. This file performs several actions:

- sets **umask** (default file access permissions) so others can read and execute but not modify or delete your files
- sets up the machine id, type, and operating system
- establishes *PATH* (by running `/usr/local/etc/setpath.sh`; see section C.2.3) and *MANPATH*.
- determines the terminal type
- sets a host of environment variables
- sets common terminal characteristics

You do not have a copy of `fermi.profile` in your home directory; it is not designed to require individual customization. See section C.2.2 for the file listing.

The `.profile` file sets your prompt and the variables that govern your history list, your default editor, and your command line editor. You can edit your `.profile` to modify your path and/or terminal settings, change the default values of variables[1] or create your own, and/or include commands that you want to execute once, at the beginning of each session (for instance **`setup`** ***`product`*** commands). See section C.2.1 for the default file listing.

.shrc and fermi.shrc

The `.shrc` file first executes `/usr/local/etc/fermi.shrc` which sets up **UPS** and performs some machine-dependent functions. See section C.2.5 for the default file listing.

The `.shrc` file should contain all your aliases[2] so that child processes have access to them; many suggested aliases are provided for you to activate, and you can define your own. You can also set variables (*noclobber* and *ignoreeof* are already set for you except in **sh**) and parameters that are local to a shell, and you can activate and define functions. See section C.2.4 for the default file listing.

Execute files to modify current session

If you modify your `.shrc` or `.profile` files and you want them to take effect in the current session, you must execute them with the `.` command:

```
$ . .shrc
```

```
$ . .profile
```

This is explained in section 4.4.

---

1. Remember for **sh**, there is not really a difference between shell and environment variables; see section 9.1.
2. Aliases are available for **bash** and **ksh**, but not for **sh**; see section 9.3.2.

9.4.3 Storing Customized Code

If you wish to maintain versions of distributed code customized to your own needs, we recommend that you store them in the following directories:

`$HOME/bin`	for machine-neutral code
`$HOME/bin.$ARCH`	for architecture-specific code; `$ARCH` is the value returned by **`funame -s`** (e.g., SunOS, IRIX).

The path names for these directories will be added to your *PATH* when the Fermi files are invoked.

9.5 X Terminal Support

A very helpful collection of information regarding X terminal configuration and use is maintained on the Web. From the Computing Division home page, select *X-terminals* under the heading **Systems and Networking**.

In this section we provide only a brief overview of the configuration and start-up information.

9.5.1 Configuration

If you have a new X terminal, you'll first need to configure it. The Computing Division currently provides boot/configuration/font service for NCD and Tektronix X terminals. There are two types of boot service, basic and complete. Basic service enables you to immediately boot the terminal, manage windows, reach the network, and have a complete set of fonts for most applications. Complete service includes the *Save Configuration* capability. Complete service requires a special request form. Refer to the Web pages for instructions.

Upon reboot, the terminals are by default loaded with a basic server and configuration. The server contains local telnet, setup, and console clients as well as two window managers. The configuration contains font service, access to the FNAL name servers, color tables, and other basic X terminal needs.

The service is provided on FNALU, but in practice, anyone with an NCD or Tektronix X terminal can use the service. In particular:

New users	If you have your terminal and there is no boot software available on your local nodes, you should use the central boot service.
Users in small groups	Groups with a small number of terminals may prefer to use the service instead of getting their own copy of the software.
Special projects	Projects which will be using the terminals for a short amount of time or without a particular host computer should use the service.
Users whose own server is down	For temporary use.

9.5.2 Connecting to Host Computers

For an NCD X terminal:

1) Select *Start Terminal* from pulldown menu on background

2) Type in host name

3) Login to host

For a Tektronix terminal:

1) Select *Telnet* From "Host Connections" menu in *Launcher*, or select or type in host name in *TekHostMenu*

2) In telnet window, type **`open`**

3) Login to host

For either type of X terminal, in order to run an X client program, you need to set your *DISPLAY* environment variable (see section 9.2).

9.6 Multimedia File Support

Applications such as Web browsers and mail handlers need to be able to handle files of many different types. The standard used for identifying multimedia file types is called Multipurpose Internet Mail Extensions (MIME).

When a server sends a document to a client, it usually includes a section that identifies the document's type so that the file can be presented properly. The identifier is called a MIME type, and consists of a general type (e.g., text, image, application, audio, video) and a subtype which specifies the format. These two elements are separated by a slash. Examples of MIME types are:

```
text/plain
text/html
image/jpeg
image/gif
application/postscript
```

A file called `.mailcap`[1] is used to map MIME types to external viewer programs, thus providing a recipe for displaying/playing multimedia files. When you first run **`setup www`** in a FUE environment, a default `.mailcap` file gets created in your `$HOME` directory if it doesn't already exist. If an earlier version of the file is found, the terminal displays a message saying that you can update it from the file in `$WWW_DIR/lib/TypeMap`. For most situations, the `.mailcap` file should be sufficient as provided.

Each entry in the `.mailcap` file consists of two fields separated by a semicolon (`;`). The first field is the MIME type in the format `type/subtype`. You may see asterisks used as wildcards to specify all of the subtypes of a particular type (e.g., `video/*`). The second field specifies the display command. It requires a full shell command, including the pathname for the external viewer and any command line arguments. Some examples of entries from the `TypeMap` file are:

```
image/xwd; display %s
image/x-xwd; display %s
image/x-xwindowdump; display %s
audio/*; sfplay %s
```

1. Its name refers to the fact that it was originally designed for multimedia mail, however its role has since expanded, and will probably continue to do so as more and more programs become multimedia.

```
video/mpeg; mpeg_play %s
video/*; animate %s
application/postscript; ghostview %s
application/x-dvi; xdvi %s
application/pdf; xpdf %s
```

The `%s` is a **printf**-style parameter for the string representing the filename.

Sometimes, if the MIME type is not sent in the file's header, the multimedia application displaying it needs to determine the file's MIME type from its file extension. In this case, the application references a file called `mime.types` which provides the mapping between file extensions and MIME types. This file is usually not required, and in fact a default `mime.types` file is not even provided.

To add support for a new MIME type with an associated file extension, you would need to create a `mime.types` file to provide the file extension mapping, and then edit your `.mailcap` file to include an entry that maps the new MIME type to an external viewer that can display the data.

For example, say you want to add support for MIME type `application` with (fictional) subtype `xyz`. The files come with the extension `xyzz`, viewable via the program **viewxyz**. You would need to create `mime.types` and include the following line in it:

```
application/xyz xyzz
```

Note there is no semicolon (`;`) in a `.mime.types` entry. Then in `.mailcap`, you would need an entry as follows:

```
application/xyz; viewxyz %s
```

More information on the `.mailcap` and `mime.types` files can be found on the Web under the heading **WWW** from the *UNIX Resources* page.

9.7 Terminal Characteristics

You can specify your terminal type to UNIX if the default is not suitable. To do so, enter the command for the C shell family:

```
% set term=termtype
```

or for the Bourne shell family:

```
$ TERM=termtype; export TERM
```

where ***termtype*** is the name of a terminal type supported on the system. vt100, vt220 and xterms are acceptable terminal types. If you always use the same kind of terminal, you may want to put this command in your `.login` or `.profile`. Note that the standard Fermi files attempt to set this variable correctly.

In Section 2.4 we listed some terminal control functions. Recall that you can display the settings with the **stty** command:

```
% stty -a
```

The format on each machine is different but should indicate approximately the same information. The following is the output from a Silicon Graphics workstation. The settings reflect the FUE defaults.

```
speed 9600 baud; line = 1;
intr = ^C; quit = ^; erase = DEL; kill = ^X; eof = ^D; eol = ^@; swtch = ^Z
lnext = ^V; werase = ^W; rprnt = ^R; flush = ^O; stop = ^S; start = ^Q
-parenb -parodd cs8 -cstopb hupcl cread clocal -loblk -tostop
-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr icrnl -iuclc
ixon ixany -ixoff
isig icanon -xcase echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3
%
```

In this display the second and third lines display the FUE default control characters. The character **^** indicates the control key (e.g., **^C** represents <Ctrl-c>). Your reference books will most likely tell you to delete a character with the # key and delete a line with the @ key, **but this is not correct at Fermilab**. Use the character indicated as erase in the **stty** output for single character deletion, and **kill** for whole line deletion. The Fermi UNIX Environment defaults for these operations are the delete key and <Ctrl-x>, respectively.

You can display a description of all of the options reported by **stty** with the command:

```
% man stty
```

If you don't like the FUE defaults, you can also set these functions with the **stty** command. The form for setting them is:

% stty ***control-char c***

where:

control-char	is one of the functions in the table in section 2.4.
c	is the representation of the key to be used for that function. A control character is specified preceded by a caret: **^x** represents <Ctrl-x>.

Example:

```
% stty kill '^y'
```

There are two special representations: **^?** is interpreted as the delete key and **^-** is interpreted as undefined. You must include the quotes as shown in the example so that special characters are not interpreted incorrectly. You must be careful not to have two functions represented by the same key.

There are many other options that can be set with **stty**. Others that might be of interest are **echoe** which specifies that deleted characters are erased, and **-tabs** which specifies that the tab character be translated into the appropriate number of spaces. Refer to the man pages for more information.

Chapter 10: Accessing Software Products

In this chapter you will learn how to get information about the software products that are provided and supported by the Computing Division. We describe how to access products already installed on your system, and how to obtain a product from the Fermilab `KITS` area and install it on your system.

Appendix B discusses the Fermilab UNIX Product Support structure, **UPS**. We recommend that you read it first to get a good basic understanding of how products are handled. At a minimum, you can refer to it to clarify the terms and concepts used in this chapter.

The information in this chapter and Appendix B has been taken from *UNIX Product Methodology at Fermilab: Guide to Using UPS v3 and UPD v2 for Product Maintenance, Installation, Distribution and Development* (GU0014), but covered here in much less detail, appropriate to the different audience. We refer you to that document if you need further information.

Notice of Upcoming Changes

UPS and **UPD** are currently undergoing redevelopment with a significantly different design. The new versions and accompanying documentation are due for release in the first half of 1998.

Off-site machines need to register for the product distribution services, **UPD** and anonymous **ftp**, discussed in sections 10.3 and 10.4, respectively. To do this, the responsible party for the machine needs to fill out the *UNIX Product Distribution Request Form* and mail it to *compdiv@fnal*. To find this form, go to the Computing Division home page, and under the heading **Services** click on *Forms*. Or look in the top level anonymous **ftp** directory (see section 10.4).

10.1 Finding Information about Available Software

The ever-changing list of software provided and/or supported by the Computing Division is maintained in a database accessible via the Web, as described in section 1.6.1. Here you can search for information on any **UPS** product, in particular the product name and number[1], an abstract describing the product, flavors for which it is available, and which versions are available in `KITS`. In addition, documents associated with the product are provided in HTML, PostScript, and/or ASCII text formats, as available.

An on-line report is available that lists the current version of all the **UPS** products. From the Computing Division home page, select *UNIX Applications* under **Documentation & Software**, and then select *Kits Report*.

1. This is an alphanumeric designation assigned by the Computing Division to identify each product and its associated documents.

10.2 Accessing Installed UPS Products

10.2.1 Get Information About Products Installed on Your System

First find out if multiple databases are defined on your system for your use by running the command:

```
% echo $PRODUCTS
```

Note the capital letters; *PRODUCTS* is an environment variable set by **UPS**.

If there is only one value, you can find out what **UPS** products are installed on your system with the command:

```
% ls $PRODUCTS
```

If the variable *PRODUCTS* is set to multiple values, you can check each directory (i.e. each database) separately.

To find the current version of a **UPS** product and its location, use the command:

```
% ups list [-a] [product]
```

More than one version, or instance, of a **UPS** product may be installed on your machine. If `-a` is specified, all instances of the product(s) are listed. If ***product*** is omitted, all **UPS** products are listed.

10.2.2 Setup a Product Instance

Each installed, declared **UPS** product instance requires that the **`setup`** command be issued prior to use (unless it is a dependent product of one that is already setup)[1]. At a minimum this command sets the environment variable *${PRODUCT}_DIR* (e.g., *WWW_DIR*, *EMACS_DIR*) to point to the root directory of the product instance. If the product has a setup file maintained in `${PRODUCT}_DIR/ups/` (most products do), **`setup`** also sources this file which makes the appropriate changes to your software environment in order to make the product available for use.

Typically users need to setup a chained instance of a product, for example the current instance on their machine. To setup the current instance of **www**, enter:

```
% setup [-c] www
```

The current instance is, in other words, the default (`-c` is unnecessary). To setup any other chained instance, include the chain flag (described in section B.10) in the command. For example, to setup the instance of **www** declared as test, enter:

```
% setup -t www
```

To setup an unchained instance of a product, include its version number. For example, to setup version 2_6a of **www**, enter:

```
% setup www v2_6a
```

1. There are exceptions to this rule; in particular, you should *never* setup the core FUE products **futil**, **fulib**, **funkern** and **shells**. **UPS** gets setup for you under FUE.

We recommend that you *not* specify **UPS** products by their version in general, but instead use primarily the *current* version for production, the *test* version (or new or development) for testing, or an *old* version *only if necessary.* We cannot maintain all versions forever, and unless there are overwhelming reasons, old versions will be removed in a timely manner.

Note that you can only have one instance of a product setup at a time. Each time you run **`setup`** on an additional instance of the same product, the previously active instance is automatically unsetup first.

Examples

```
% setup prod1
```

Sets up the current instance of product **prod1** and all of **prod1**'s use requirements.

```
% setup -d prod1
```

Sets up the instance of product **prod1** chained to development, and all of **prod1**'s use requirements.

```
% setup -v -f IRIX prod1 v1_1
```

Sets up the instance of **prod1** that has a version of v1_1 and a flavor of IRIX, and all of **prod1**'s use requirements. Sets verbose on.

10.2.3 Unsetup a Product Instance

`unsetup` sources the unsetup script, also maintained in `${PRODUCT}_DIR/ups/`. In general this undoes the changes to your software environment made by **`setup`** in order to make the product no longer available for use. You may need to unsetup a **UPS** product if you are short on environment variable space and want to get rid of extra environment variables, or shorten the *PATH* variable length.

When you no longer need to access a product (or its use requirements), simply type:

```
% unsetup product
```

10.2.4 Invoke the Product

Normally **UPS** products are invoked by entering the product name on the command line, with any appropriate options and/or arguments, e.g.,

```
% setup prod1
% prod1 [options]
```

Note that you must know the name of the application or its invoking command, **which in a few cases is different from the UPS product name used in the `setup` command.** If the product comprises multiple applications, you can usually get a listing of them by checking the product's `bin` directory. After you setup the product, run:

```
% ls ${PRODUCT}_DIR/bin
```

For example, say you want a tool to format a document in PostScript. First you need to setup the **UPS** product **psutils**, which includes all the available tools. Then run the command:

```
% ls $PSUTILS_DIR/bin
```

to see what your choices are:

```
a2ps          fixmacps     fixtpps      includeres   psmerge      showchar
epsffit       fixntps      fixwfwps     psbook       psnup
extractres    fixpsditps   fixwpps      pscover      psresize
fixdlsrps     fixpspps     fixwwps      psduplex     psselect
fixfmps       fixscribeps  getafm       pslabels     pstops
```

To invoke one of them, just enter the name of the executable on the command line. Be aware that for some products, files other than executables may also be included in `bin`.

10.3 Obtaining Products from KITS

The **UPD** menu interface was designed for easy distribution of software products, either directly via the network or by an intermediate tar file stored on disk or tape. Tar files of the currently available versions of **UPS** products for the supported UNIX flavors are maintained in the Fermilab `KITS` area on the node UPD.FNAL.GOV. **UPD** provides functionality to:

- list products in `KITS` or any other **UPS** product distribution node
- list the contents of a remote tar file
- download a product instance or copy an individual file from a remote tar file

UPD can also be used to:

- unwind the tar file once it's on the local node
- declare the product instance to the **UPS** database

As a user, you should be able to largely rely on your system administrators to install **UPS** products for you. Occasionally you may need a product that has not been installed on your machine.

If you want to make a **UPS** product available for general use on your system, you should have it added to your local system's **UPS** database. Contact your system administrator in this case.

If a product is for personal use, we recommend that you download the tar file to a temporary area using the **UPD** function provided, and unwind the retrieved tar file into your personal area. Be careful that you have enough disk space available.

Recall that off-site systems must register for **UPD**.

10.3.1 Steps for Installing a Product

Before installing any product:

1) Create at least the directory in which the product root directory will reside; you can create the product root directory itself, too, but it is not necessary.

2) (optional) Change to the product root directory or to the directory in which it will reside.

3) Run **`setup upd`**.

4) Run **`upd`** to invoke the **UPD** menu interface.

You should see the following main menu (version v2_8d of **UPD** was used here):

```
Welcome to the main menu interface for upd.
      Choose one of the following options:

        1       upd list: list available products
        2       upd copy: copy a product from a remote node
        3       upd unwind: unwind product into local directory
        4       upd tarcont: list the table of contents of the tar file
        5       upd extract: extract a file from tar file
        6       upd getreq: determine the build and use requirements
        7       ups declare: declare the product to UPS
        8       ups tailor: execute product's ups/tailor file
        9       fork a login shell

        q       quit
Enter choice :
```

Products are installed in a maximum of six steps, all of which can be accomplished via this menu:

1) List available products from the remote distribution node, and determine which product instance to install (option 1).

2) Copy the tar file associated with the chosen product instance to a temporary tar file on the local node (option 2).

3) Unwind the temporary tar file into its product root directory (option 3).

4) Declare the product to the **UPS** database (option 7), with or without a chain.

5) Tailor the product if **UPD** informed you that it is necessary (option 8).

6) Declare the product as "current" to **UPS** (option 7, again), if not already done.

Note that **UPD** also allows you to download individual files from within a remote tar file. If, for example, one or two files of a previously installed product get corrupted, you can use UPD TARCONT to list the contents of the tar file and UPD EXTRACT to download an individual file from it.

A few notes on installing products:

- If additional actions are required by the installer, the product's `INSTALL_NOTE` is automatically typed to the screen during the unwind process to inform you about them.
- Very few products require tailoring. **UPD** informs you during the declaration if tailoring needs to be run.
- If you are using a version of **UPS** previous to v3_9, we do not recommend simultaneously declaring a product to the **UPS** database and declaring it current (steps 4 and 6). This is due to a slight design flaw that is corrected in later versions (described in GU0014).
- If the product has use requirements (you shouldn't need to worry about build requirements in most cases), you are informed of this fact during the unwind step. Use option 6 to find out what the requirements are. You will need to install each required product instance individually if it does not already exist on your system.
- According to your group's policy, communicate to the users of your system if you make a new product instance available for their use.

10.3.2 UPD Menu Interface Operations

Each step of the installation has a screen associated with it, similar to the sample screen below:

```
UPD LIST:   * implies value required
        *1      [upd.fnal.gov] remote node name
         2      [] product name
         3      [current] version or chain (e.g., v1_0 or current)
         4      [SunOS+5] flavor
         5      [] remote database, if different from default

         e      execute upd list
         q      quit and return to main menu
 Enter choice :
```

The screens each have a list of numbered parameters to set. Defaults are provided where possible, and you can change or remove them. If you exit **UPD** and then restart it, you will lose any values associated with the particular product instance you were working with. The option of forking a shell from within **UPD** is provided in order to avoid this problem.

To set a parameter, enter its number at the `Enter choice :` prompt, and press carriage return. You set the parameters individually in successive operations. The parameters labelled with an asterisk (*) require you to set a value. All other parameters are optional. When an optional parameter does not have a value, the screen function generally executes for all possible values.

Once you've set all the parameters that you want, enter **`e`** to execute the screen function. Press **`q`** to quit the screen and return to the main menu. If you're on the main menu, pressing **`q`** quits out of **UPD**.

10.4 Using Anonymous ftp to Download a Product

UPS products are also available via anonymous **ftp** from the host FTP.FNAL.GOV. We recommend that after copying over the product tar file using **ftp** you use **UPD** to install the product.

If you have a local **UPS** database, we recommend distributing via **UPD** because it provides a better interface for declaring products once the tar file is copied. You can always invoke **UPD** to do this after downloading via **ftp**, of course.

10.4.1 Access Anonymous ftp

To access anonymous **ftp**, enter the command:

```
% ftp ftp.fnal.gov
```

You will get a response similar to the following, to which you enter **`anonymous`** at the `Name` prompt:

```
Connected to fsui01.fnal.gov.
220 fsui01 FTP server (Version wu-2.4(2) Thu Jul 20 12:03:52 CDT 1995) ready.
Name (ftp.fnal.gov:your_username): anonymous
```

Once that is accepted, you will be prompted to enter your email address as a password (e.g., `qjones@fnal.gov`), which is not echoed to the screen:

```
331 Guest login ok, send your complete e-mail address as password.
Password:
```

The screen will then display the following information[1]:

```
230-This is the FTP service for FTP.FNAL.GOV. Use of this
230-service is for authorized usage only.
230-
230-The purpose of this service is to provide access to the
230-following categories of files:
230- - The FermiTools area: publicly available software under the
230-   /pub directory (all users). Files available listed in ls-lR.Z.
230- - The UNIX KITS area: This provides alternate access to files
230-   normally accessed via the upd program. (registered users)
230-   Files available listed in index.
230-
230-This server allows for on-the-fly compression  and uncompression
230-of files retrieved. It also allows you to tar up entire directory
230-structures.
230-
230-You are registered to retrieve files from the UNIX kits area.
230-
230-Please read the file readme
230-  it was last modified on Wed Sep 27 16:47:32 1995 - 415 days ago
230 Guest login ok, access restrictions apply.
```

In the top level directory a `readme` file is provided that discusses some of the available tools and features.

If you are on an off-site node, it must be registered with Fermilab in order to use anonymous **ftp** to obtain tar files[2] from the `KITS` area. If you need to register, use the UNIX Product Distribution Registration Form in the top level directory under the filename `registration`. All machines in the `fnal.gov` domain are automatically registered.

1. The "last modified" information will change, of course.
2. Note, the FermiTools products are not subject to this restriction; you do not need to register your node to obtain tar files from the FermiTools `/pub` area.

Here is a top-level directory listing for reference:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 66
dr-xr-xr-x   8 ftp      root        1024 Nov 19 22:00 .
dr-xr-xr-x   8 ftp      root        1024 Nov 19 22:00 ..
drwxrwxr-x   7 ftp      995          512 Oct 31 11:21 .incoming
drwxr-xr-x  20 kits     sys         1024 Nov 19 22:00 KITS
lrwxrwxrwx   1 root     other          7 Jul 27  1995 bin -> usr/bin
dr-xr-xr-x   2 root     other        512 Jul 27  1995 dev
dr-xr-xr-x   2 root     other        512 Apr 24  1996 etc
drwx------   2 root     root        8192 Jul 27  1995 lost+found
-rw-r--r--   1 root     other      11824 Nov 19 22:00 ls-lR.Z
lrwxrwxrwx   1 root     other          8 Sep 15  1995 pub -> KITS/pub
-rw-r--r--   1 root     other       1566 Sep 27  1995 readme
-rw-r--r--   1 root     other       1937 Mar 26  1996 registration
dr-xr-xr-x   5 root     other        512 Apr 24  1996 usr
226 Transfer complete.
827 bytes received in 0.17 seconds (4.6 Kbytes/s)
```

Most **UPS** products can be found under the `KITS` directory. Access to this directory is restricted to registered users. Directly under `KITS` is a directory for each supported OS type, under which the product instance tar files appropriate to each one are maintained. The `declared` directory containing the product declaration files is also available under `KITS`. The public **ftp** area `KITS/pub`, available to all (registered and nonregistered) users, includes the Fermilab Software Tools (FermiTools). This collection of software is intended to provide the internet community with many of the Fermilab-developed software packages that we believe have general value to other application domains.

To see the contents of `KITS`, run the following (the output has been abbreviated for this manual):

```
ftp> cd KITS
250 CWD command successful.
ftp> dir
drwxr-xr-x 102 kits     sys         2048 Oct 29 17:31 AIX
drwxr-xr-x  53 kits     sys         1024 Nov 19 10:53 GENERIC_UNIX
drwxr-xr-x  25 kits     sys          512 Aug 15 13:57 HP-UX
drwxr-xr-x 153 kits     upd         2560 Oct 29 17:38 IRIX
drwxr-xr-x   2 kits     sys          512 Aug 27 14:47 LINUX
drwxr-xr-x  84 kits     upd         1536 Nov  6 13:29 OSF1
drwxr-xr-x 111 kits     sys         2048 Oct 29 17:46 SunOS
drwxr-xr-x  41 kits     sys         1024 Aug 15 13:56 ULTRIX
drwxr-xr-x   2 kits     sys         3584 Nov 19 12:08 declared
drwxrwsr-x  22 bin      frmtools     512 Nov 14 22:29 pub
```

10.4.2 Select a Product Instance Tar File

From the root directory (/), **cd** to the appropriate OS type subdirectory under KITS (SunOS is used as an example here). You'll need to "get" the index file since you can't list a remote file from **ftp**. Since index is an ASCII file, make sure you set the mode to ASCII first, as shown below:

```
ftp> cd KITS/SunOS
250-A complete listing of files available under this directory
250-structure is available in the index file.
250-
250 CWD command successful.
ftp> ascii
200 Type set to A.
ftp> get
(remote-file) index
(local-file) index_SunOS
200 PORT command successful.
150 Opening ASCII mode data connection for index (74193 bytes).
226 Transfer complete.
local: index_SunOS remote: index
76388 bytes received in 0.89 seconds (84 Kbytes/s)
```

Now you have a local (renamed) copy. Use the **ftp** command **!** to run **less** as a shell command to list the file contents:

```
ftp> !less index_SunOS
```

You'll find that the file lists everything under KITS; all directories, subdirectories and tar files.

10.4.3 Copy the Tar File

Any file that is not a straight text file must be transferred in binary mode. Set the mode to binary (**bin**) and then use the standard **ftp get** function to copy the tar file. Refer to section 13.1.1 for information on standard **ftp** commands.

After the tar file is downloaded, use **UPD** to unwind it into the appropriate location on your system and declare it, as described in section 10.3.1.

Chapter 11: Editors

Several text editors are available at Fermilab. In this chapter we present our view of the advantages and disadvantages of the available editors, and we provide some basic information on the setup and use of each one. You will learn how to invoke each editor, and how to create, edit, and save a file in each one using a small subset of commands and features. We include only minimal usage information for the VMS-style editors.

11.1 The Available Editors

vi	A native UNIX screen editor. It is not generally considered to be one of the better available editors, however knowing the basics is useful for two reasons: 1) you may occasionally encounter an application that throws you into a **vi** session, from which you'll at least want to exit, and 2) **vi** is the one editor you are guaranteed to find on all UNIX machines[1].
emacs	A powerful modern public-domain screen editor available for both VMS and UNIX. You will find **emacs** installed on most UNIX machines.
xemacs	An incarnation of the advanced, self-documenting, customizable, extensible real-time display editor **emacs**. It provides many powerful display and user-interface capabilities not found in **emacs**.
NEdit	A popular and intuitive X-based editor written at Fermilab for UNIX.

The remaining available editors are UNIX implementations of Digital's products for VMS:

nu/TPU	An emulation of TPU
fermitpu	A locally ported UNIX version of the newest EVE, layered on **nu/TPU** (from VMS V6.1)
EDT+	A UNIX version of EDT[2]

The VMS-style editors are legacy products, and as such are not guaranteed to remain available beyond the VMS-to-UNIX migration period. It is also unlikely that you will find these third party editors on UNIX systems outside of Fermilab. They are provided solely to ease the transition to

---

1. In addition, UNIX provides **sed** for non-interactive editing and **awk** for more sophisticated and complex non-interactive editing. These two products are not covered in this manual.
2. Due to budgetary constraints and soaring maintenance costs, support for **EDT** under UNIX is now frozen at release v6_3a. We expect that this release will continue to function properly. However, if future hardware or software changes cause **EDT** to break, it will not be replaced or upgraded. **EDT** will *not* be purchased for any new operating systems which may be supported in the future.

UNIX for users already familiar with these products on VMS. **Users not already familiar with these VMS editors are strongly discouraged from choosing one of them for use in UNIX!** *All* users are encouraged to begin learning one of the permanent UNIX editors.

11.2 Comparison of Editors

The following table discusses the advantages and disadvantages of each of the above-mentioned editors.

Editor	Advantages	Disadvantages
vi	Available on all supported platforms and systems as a native UNIX tool. Well-documented. It is set as the default editor for many applications on many systems.	Considered to be very primitive in its screen capabilities. Non-intuitive interface. Lowest common denominator of screen editors.
emacs	Widely available public domain software package. Well documented. Many macros and enhancements available via the internet. Language-specific text editing modes (e.g., English, Lisp, C, FORTRAN). Lots of expertise and consulting available via newsgroups, etc. Very configurable. **EDT**-style keypad emulation available. X and ASCII modes. Supports file versions.	Requires installation of **emacs**. Interface not very intuitive for users accustomed to VMS-style editors.
Xemacs	Same as for **emacs**, plus more. Intuitive, GUI interface. Language-specific syntax-highlighting. Multi-windowed, interactive, object-oriented class browser.	Requires installation of **Xemacs**.
NEdit	Very intuitive and well-documented. Customizable. Gaining popularity world-wide. X-based.	Requires installation of **NEdit**. Requires X-based terminal. Not available on all systems/platforms.
nu/TPU	Nearly identical to Digital's **TPU** for people familiar with **TPU** programming. Sufficient for a wide variety of simple editing tasks. Close enough to **EVE/EDT**-style keypad to be familiar to VMS users. Does not exhibit screen-repainting problem of **fermitpu** (see below).	Duration of availability is uncertain. Implementation includes many faults. Limited documentation available on-line. Uses old version of **EVE** (from VMS V5.0). Does not match the interface that VMS users are used to. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.

Editor	Advantages	Disadvantages
fermitpu	Nearly identical to Digital's **EVE** of VMS V6.1 with the exception that the **EDT**-style keypad is used by default. Licensing is handled on the local node (i.e., no network license server problems). User-configurable and extendable. Does not require X.	Duration of availability is uncertain. Not available/installed on all systems/platforms. Not the same as **fermitpu** on the central facility VMS systems, although similar. Requires the FUE environment and the installation of **nu/TPU** and the **fermitpu** layered products. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings. On some platforms, the screen is repainted after every keystroke which is very slow.
EDT+	Very similar to an enhanced **EDT** for people familiar with **EDT** programming. Incorporates a powerful word processor, disaster recovery system, an extensive help facility, full feature programmable text processing and the GOLD-KEY style of editing. Does not require X, although X interface is available. Minimum learning curve for users familiar with Digital's **EDT**.	Duration of availability is uncertain; maintenance contract terminated. Requires the FUE environment and the installation of the **EDT** product. Not available/installed on all systems/platforms. Distributed licensing (implies that network problems and/or problems with the license server could prevent **EDT+** from being usable on occasion). Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.

Note: For all editors used with X display, the *DISPLAY* variable (see section 9.2) must be set properly.

11.3 Getting Started with the Editors

In this section we present a few important commands for each editor. The minimal information necessary for you to edit and save a simple file is provided. All these editors have many commands and sophisticated features that we do not cover here.

11.3.1 vi

As we mentioned in section 11.1 above, you may occasionally find yourself thrown into the **vi** editor unexpectedly, and you will certainly want to know how to exit, if nothing else. For that reason alone you should become familiar with a few **vi** commands. As is typical in UNIX, the **vi** commands are case sensitive.

vi requires no setup. Invoke it using the command:

```
% vi [filename]
```

Once the file is opened, you are in *command mode*. From this mode you can issue commands, move the cursor, and invoke insert mode.

To enter *insert mode*, type `i`. Text you type will be inserted *before* the cursor. From insert mode you can enter new text in the file. Press the Escape key to exit insert mode and return to command mode. On many keyboards the Escape key is labelled; if not, the sequence <Ctrl-[> is mapped to the escape function.

Some useful vi commands available in command mode

`h,j,k,l`	move cursor left, down, up, right, respectively
`H`	move to top line of screen
`L`	move to bottom line of screen
<Ctrl-f>, <Ctrl-b>	scroll forward, backward one screen
/pattern	search for ***pattern***
`/`	repeat search in forward direction
`x`	delete current cursor position
`X`	delete back one character
`dw`	delete current word
`dd`	delete current line
***n*`dd`**	delete ***n*** lines starting with current
`p`	insert (paste) last deleted text after cursor
`:r` ***filename***	read in contents of ***filename*** after cursor
`:x`	quit **vi**, writing file only if changes were made
`:w`	write file, do not quit
`:w` ***file***	save copy to ***file***, do not quit
`:q!`	quit file, discarding edits

Most UNIX guides contain a complete description of **vi**.

11.3.2 emacs and xemacs

We'll cover these two editors in the same section because although they are separate products, they are closely related.

emacs is a popular editor available on the net. It can be invoked in windows mode or ASCII mode. Many UNIX books cover **emacs**, and a good reference for **GNU emacs** is *Learning GNU Emacs* (O'Reilly & Associates). We also refer you to an on-line GNU EMACS manual accessible under **Editors** on the *UNIX Resources* Web page. Further documentation can be found from the man pages.

xemacs is a graphical, X window implementation of **emacs**, with a few extra bells and whistles. (It is somewhat fancier than **emacs** in windows mode, and not to be confused with it.) **xemacs** is very similar in appearance and operation to many PC text processing applications for Windows. It is menu/mouse driven, with keyboard shortcuts available. **xemacs** has commands for passing single command lines to shell processes; it can also run a shell interactively. All standard **emacs** keyboard commands can be used in **xemacs** instead of/in addition to the pull-down menu commands. An **xemacs** manual is accessible under **Editors** on the *UNIX Resources* Web page.

Setup and Invoke emacs

To use **emacs**, the product needs to be installed. To set it up, include in your login script or enter:

```
% setup emacs
```

The mode (X or ASCII) in which **emacs** attempts to start-up is determined according to your *DISPLAY* variable. To invoke, type:

```
% emacs [options] [filename]
```

To invoke it in ASCII mode without a new window using an X terminal, type:

```
% emacs -nw [options] [filename]
```

Setup and Invoke xemacs

To use **xemacs**, the product needs to be installed. To set it up, include in your login script or enter:

```
% setup xemacs
```

To invoke **xemacs**, type:

```
% xemacs [filename] [&]
```

Help Facilities

emacs/xemacs has an extensive interactive help facility, but the facility assumes that you know how to manipulate **emacs** windows and buffers. <Ctrl-h> enters the Help facility. <Ctrl-h>-t enters the help tutorial, which can teach beginners the fundamentals of **emacs** in a few minutes. <Ctrl-h>-a enters Help Apropos, to help you find commands by function. For **emacs** in windows mode, there is also a Help menu. <Ctrl-h>-i enters the Info facility which brings up the on-line documentation browsing system. The initial page (the Directory node) gives a menu of major topics. The information is presented in a hierarchical tree format. **xemacs** provides this via an Info button.

Keyboard Commands

emacs/xemacs commands use the *Control key* and the *Meta key*[1]. In the following list `C-` indicates that the control key is pressed at the same time as the character that follows. Similarly, `M-` indicates the use of the Meta key, although it's not necessary to keep the Meta key pressed down while typing the next character. Note that some command sequences use multiple keystrokes, with and without the Control and Meta keys. A sequence like `C-x u` means hold down control while you press `x`, then just press `u`. Following is a list of the **emacs** commands used most often:

`C-h`	enter on-line help system
`C-h i`	enter the information browser which provides a menu of major topics (use tab and enter keys or 2nd mouse button to select a topic; navigation information is provided)
`C-p`, `C-n`, `C-f`, `C-b`	move up (to previous), down (to next), forward, or backward by one line or character, respectively
`C-v`, `M-v`	move forward, backward, by one screen
`C-s`	search forward for characters (system will prompt you for string). To continue search, type `C-s` again.
`C-r`	search backward

1. If you have a key labelled Meta, use it; if you don't, try the Alt key or the Escape key (if you're running a native X window). As a last resort, the sequence <Ctrl-[> should always work as a Meta key.

C-d	delete a character
C-k	delete (kill) from cursor to end of line
C-y	restore what you've deleted
C-x u	undo last edit
C-g	get out of current command operation
C-x i	insert file at cursor position (system will prompt for filename)
M-q	fill paragraphs
C-@ or **C-spacebar**	set the mark for the start or end of a region to select
C-w	delete all between mark (see **C-@**) and cursor's current position (paste back with **C-y**)
M-w	copy all between mark (see **C-@**) and cursor's current position (paste back with **C-y**)
C-x C-x	exchange mark and cursor's current position (since the mark is invisible, this allows you to find it)
C-x C-s	save the file
C-x C-w	write to file (system will prompt for filename)
C-x C-b	display buffer list
C-x o	move cursor to other window (when more than one displayed)
C-x C-c	exit emacs

Note, if the serial port or terminal device you are typing on is configured for <Ctrl-s>/<Ctrl-q> flow control, you may find that <Ctrl-s> (written above as **C-s**) within **emacs** causes the terminal to stop sending characters, the same as when used at the shell prompt. If you want to use the usual **emacs** key bindings and to have **C-s** work properly within **emacs**, you'll need to reconfigure your line to not do flow control. Where and how you do this depends on how you're connected. If you're connected via a modem, you may need to reconfigure your modem, as well as the pseudo-terminal on your UNIX host. The latter can be done via the command:

```
% stty stop undef start undef
```

(which sets the stop and start characters to "undefined"). You could include this statement in your `.login` or `.profile`. The intermediate step, between the on-site modem and the Cisco router/terminal server, has flow control turned off by default.

Language-Specific Text Editing Environments

emacs/xemacs supports several text editing environments (called modes), each geared to a particular language (e.g., English, Lisp, C, FORTRAN). When the editor is started, it normally loads the file `$HOME/.emacs`, if present, which contains Lisp commands for initialization. In particular, by setting up a mapping in this file between file extensions and languages, you can configure **emacs/xemacs** to come up in the appropriate mode according to the extension of the file you specify on the invoking command line. The text you need to include in `.emacs` has the syntax:

```
(setq   auto-mode-alist (append '(
    ("\\.extension1$" . language1-mode)      ("\\.extension2$"   . language2-mode)
    ("\\.extension3$" . language3-mode)      ("\\.extension4$"   . language4-mode)
    ...
      ) auto-mode-alist))
```

For example:

```
(setq   auto-mode-alist (append '(
    ("\\.asm$" . asm-mode)      ("\\.s$"    . asm-mode)  ("\\.awk$" . awk-mode)
    ("\\.cc$"  . c++-mode)      ("\\.C$"    . cc-mode)   ("\\.hh$"  . c++-mode)
    ("\\.c$"   . c-mode)        ("\\.h$"    . c-mode)    ("\\.i$"   . c-mode)
    ("\\.m$"   . objc-mode)     ("\\.csh$"  . c-mode)
    ("\\.cdf$" . fortran-mode)  ("\\.cin$"  . fortran-mode)
    ("\\.for$" . fortran-mode)  ("\\.f$"    . fortran-mode) ("\\.F$" . fortran-mode)
    ("\\.inc$" . fortran-mode)  ("\\.car$"  . fortran-mode)
    ("\\.cra$" . fortran-mode)  ("\\.crb$"  . fortran-mode)
    ("\\.tex$" . TeX-mode)      ("\\.txi$"  . Texinfo-mode)
    ("\\.el$"  . emacs-lisp-mode) ("\\.icc$"  . c++-mode)
      ) auto-mode-alist))
```

The on-line GNU EMACS manual provides more information on creating and modifying this file.

Use with EDT-Style Keypad

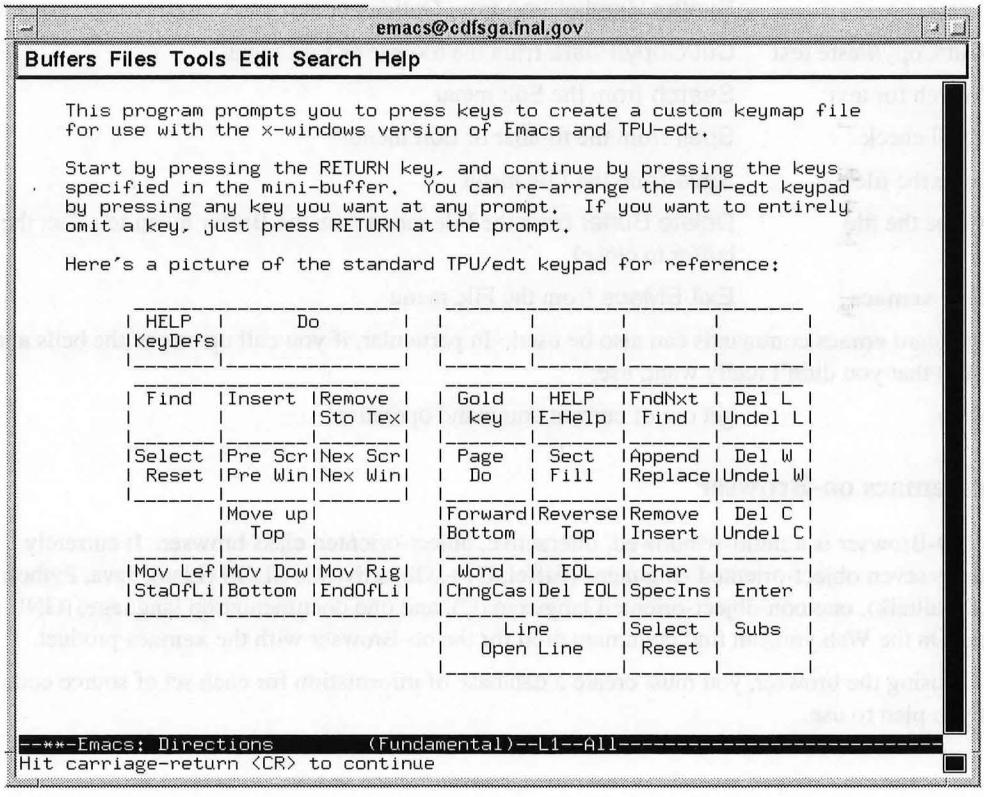
emacs/xemacs can be set to have an **EDT**-style keypad. This is documented more thoroughly in our Web pages; here we provide some start-up information.

emacs (windows) and **xemacs**

The key bindings are in the control of the editor emulation. When you first start up the **tpu-edt** emulator (instructions follow) it will prompt you to setup a keyboard mapping (see screen below). You need a separate mapping for each different keyboard type you use.

emacs in non-windows mode

Here you are at the mercy of your window emulator. There are things you can do to remap keys on most vt100 window emulators, but it is different for each OS/emulator. There are too many permutations to document.



To invoke the emulation, put the following text at the top of your `$HOME/.emacs` file:

```
(tpu-edt)               ;; Basic Emulation
(tpu-set-scroll-margins "10%" "15%") ;; Set scroll margins 10% (top) and 15% (bottom).
(load "vt-control" t)   ;; VT terminal controls (No complaint if not available)
;;
;; TPU-edt treats words like EDT; here's how to add word separators.
;; Note that backslash (\) and double quote (") are quoted with '\'.
(tpu-add-word-separators "]\\[-_.\"=+()'/*#:!&;$")
```

To try this out without changing your `.emacs` file, first invoke the editor, then press Alt-**x** (hold down Alt while pressing **x**), and type **tpu-edt** followed by a carriage return.

On some platforms you may have trouble with the Gold key. The problem and its solution are dependent on the type of terminal and keyboard you are using. We know the solutions for some combinations. We are planning to provide this information soon in a Web page under *UNIX Resources*, and to keep it updated as we learn more. You may also find some helpful information under the heading X-terminals from the CD home page; see *Misc. Information - Hints, Notes, User Experiences, Etc.*

The xemacs GUI Interface

Due to the user-friendly nature of the product, we present only a few basic commands to give you a flavor of this type of editor if you are not familiar with it:

Open a file	Open from the toolbar or File menu (you can also open in another window or in a new frame); choose an existing file from popup window
Create a new file	Open from the toolbar or File menu; type in the new file name
Include a file	Insert File from the File menu
Select text	Use the mouse
Highlight special syntax in color (for use with language modes)	Syntax Highlighting from Options menu.
Cut/Copy/Paste text	Cut/Copy/Paste from the toolbar or Edit menu
Search for text	Search from the Edit menu
Spell check	Spell from the toolbar or Edit menu.
Save the file	Save from the File menu
Close the file	Delete Buffer from the File menu (use the Buffer menu to select the buffer to close)
Exit **xemacs**	Exit EMacs from the File menu

All standard **emacs** commands can also be used. In particular, if you call up one of the bells and whistles that you didn't really want, use:

C-g	get out of current command operation

The xemacs oo-Browser

The OO-Browser is a multi-windowed, interactive, object-oriented class browser. It currently supports seven object-oriented languages (Eiffel, C++, Objective-C, CLOS (Lisp), Java, Python and Smalltalk), one non-object-oriented language (C), and one documentation language, (GNU Info). On the Web you can find documentation for the oo-Browser with the **xemacs** product.

Before using the browser, you must create a database of information for each set of source code files you plan to use.

Here is a brief set of instructions for creating a database:

1) Run **`setup xemacs`**

2) Change to the directory in which you want the browser database output to go (we'll call it `outputdir`), and invoke **xemacs**.

3) Select OO-Browser from the Tools pull down menu. In the prompting window at the bottom of the window it will say: `Load/Create OO-Browser Environment: {outputdir}/`

   Enter a filename. This file is used to store the answers to the following questions. The application will then read this file to build your browsing environment. The file can be reused in future sessions. Here we'll use the filename `OOBR`, to create the file `{outputdir}/OOBR`.

4) Next it prompts for a language: `Choose: 1) C++/C; 2) Eiffel; 3) Info; 4) Java; 5) Lisp; 6) Obj-C; 7) Python; 8) Smalltalk`

   Enter the number corresponding to your choice.

5) Some error messages may rapidly scroll by. Ignore them[1] and wait for the following prompt: `Please specify the "OOBR" Environment (Hit RET to begin).`

   Enter Return as requested.

6) Next it prompts for a list of "system" directories and then "library" directories. You can specify each directory using an absolute or a relative path name (relative to your current working directory). Specify your own source code files as "system", and any library source code files you need as "library". Terminate the list by entering a carriage return on a fresh line.

7) The next prompt is: `Build Environment from spec in file, "{outputdir}/OOBR"? (y or n)`

   Enter **`y`**

8) The final prompt is: `Build Environment in a background process? (y or n)`

   Enter **`n`** (in order to monitor what happens)

The oo-Browser starts scanning all of the files in the directory tree underneath the directories you specified. When it finishes, your database is made and you are ready to start browsing.

A couple of useful keys are f and v. f displays an expanded (full) summary of the member functions belonging to the selected class. If you enter v with the cursor on a class name, the source file that defines the class (usually a header file) is displayed for viewing. Enter e to display it for editing. If you enter v with the cursor on a member function name, it displays the source code for that member function (usually an implementation file). If there are many functions in the same file, the browser places you at the correct line number for the selected function.

---

1. A knowledgeable source suspects that these messages are the result of a bug and will go away in a future release. She has heretofore ignored them with no ill effects.

11.3.3 NEdit

NEdit is very similar in appearance and operation to many PC text processing applications for Windows. It is menu/mouse driven, with keyboard shortcuts available. Some UNIX shell commands are available from within the editor. Make sure the **NEdit** product is installed on your system. To set up **NEdit**, include in your login script or enter:

```
% setup nedit
```

To invoke **NEdit**, type:

```
% nedit [filename]
```

Due to the user-friendly nature of the product, we present only a few basic commands to give you a flavor of this type of editor if you are not familiar with it:

Open a file	Open from the File menu; choose an existing file from popup window
Create a new file	New from the File menu
Include a file	Include from the File menu
Select text	Use the mouse, or the shift and arrow keys together
Cut/Paste text	Cut/Paste from the Edit menu
Search for text	Find from the Search menu
Fill paragraph	Fill Paragraph from the Edit menu
Spell check	spell from the Shell menu
Save the file	Save from the File menu
Close the file	Close from the File menu (you are prompted about saving)
Exit **NEdit**	Exit from the File menu

Further information is available from the man pages and a plain text document in the distribution kit, but the on-line help in the program is complete and more convenient. The documentation is also available on the Web in the product documentation area.

11.3.4 nu/TPU

nu/TPU is an excellent port of Digital's TPU (Text Processing Utility) programming language to UNIX. The major component in **nu/TPU** is software that can be used to build new text processors and batch-oriented text manipulation routines. **nu/TPU** is distributed with two editing interfaces:

- SI (Simple Interface), an **EDT**-style keypad map. This should be sufficient for users who do minimal editing and wish to use an **EDT**-style keypad.
- a clone of the old **EVE** interface (from VMS V5.0) complete with the EVE_functions (**but not the EVE$functions!**) from that version. This will probably not be satisfactory to users familiar with the later version of **EVE** (VMS V6.1), since it lacks many of the enhancements and fault corrections.

The product **tpu** needs to be installed prior to use. To set up **nu/TPU,** include in your login script or enter:

```
% setup tpu
```

To edit using the SI interface by default, type:

```
% tpu [filename]
```

To edit using the older **EVE** interface, enter:

```
% tpu -section=tpusec.ini [filename]
```

Once you are in the editor, on-line help is available via the PF2 key or the Help command.

For complete documentation on **nu/TPU**, including documentation on all command-line options and all **nu/TPU** programming statements, you can order the *nu/TPU Reference Manual.* Look under **Editors** on the *UNIX Resources* Web page for documentation on the **TPU-edt** editor for GNU Emacs.

11.3.5 fermitpu

The **fermitpu** product is layered upon **nu/TPU** and contains a locally developed port of Digital's **EVE** editing interface from VMS V6.1 (the version of VMS which is current during the VMS to UNIX migration). Note that this is not the same as the VMS product **fermitpu**, which contains many enhancements beyond the VMS V6.1 **EVE** editor. Users who are familiar with programming in **EVE** and **TPU** will probably be most comfortable with this editing interface.

The products **fermitpu** and **nu/TPU** must be installed prior to use. To invoke, include in your login script or enter:

```
% setup fermitpu
```

```
% tpu [options] [filename]
```

Once you are in **fermitpu**, the on-line help (accessible via PF2 or the Help command) contains documentation on all of the EVE callbacks and TPU built-ins.

Look under **Editors** on the *UNIX Resources* Web page for documentation on the **TPU-edt** editor for GNU Emacs.

11.3.6 EDT+

Support for **EDT** under UNIX is now frozen at release v6_3a.

This is a clone of Digital's **EDT** which includes the enhancement to remove the hard-coded 24-line display limit. The product **edt** needs to be installed. To setup **EDT+**, include in your login script or enter:

```
% setup edt
```

To invoke for editing within the same terminal window, type:

```
% edt [options] [filename]
```

To edit in a separate display window (requires X), type:

```
% xedt [options] [filename]
```

EDT+ allows both line mode and full screen editing. Full screen editing employs a keypad, which is located on the right-hand side of your keyboard. If you are in line mode when you first enter **EDT+**, simply type `c` to change to full screen mode. Once in full screen mode, you can start entering text, and your keypad is available for commands.

To return to line mode, enter <Ctrl-z>. At the prompt (*) you can type `exit` to exit and save, or `quit` which exits without saving the file updates.

Remember that when you're at the UNIX command prompt <Ctrl-z> is used as the suspend character.

Further documentation is available from the man pages, and from the vendor Boston Business Computing, Ltd.

Look under **Editors** on the *UNIX Resources* Web page for documentation on the **EDT** emulation package for GNU Emacs.

Chapter 12: UNIX Mail Systems

This chapter describes how mail forwarding is managed at Fermilab and discusses the UNIX mail handlers that are currently available and supported.

Notice of Upcoming Changes

Electronic mail is an area that is currently undergoing rapid growth and development, and it is important that we take advantage of new technologies that are becoming available. In particular, demand has been growing at Fermilab for a distributed mail system that allows users to access their mail from any machine in the network, including a variety of UNIX and non-UNIX platforms.

Of the available distributed mail technologies, IMAP (Interactive Mail Access Protocol) is the one that offers the most flexibility and greatest performance. It is a client-server mail protocol designed to permit manipulation of remote mailboxes as if they were local. A particular advantage it offers over its competitors is access to remote saved-message *folders*. We are currently investigating mail handlers that support IMAP, and will announce changes within the next few months regarding the mail handlers that will be available to Fermilab UNIX users.

As of the release date of this manual, we are providing and supporting the same mail handlers as previously documented, namely **MH** (including **exmh** and **mh**) and **pine**. We plan to continue providing them after IMAP is implemented. However, new users should think about whether they will eventually want to move to an IMAP-based mail handler *before* choosing **MH** or **pine**. They should also be aware of a couple of disadvantages associated with the **MH** products:

MH does not and never will support IMAP. Furthermore, the **MH** suite of products is no longer under development, at least partly because its underlying structure doesn't support any of the emerging distributed mail technologies. The future of **MH** is therefore uncertain. It is unlikely that we will continue to build more versions of **MH** for OS upgrades once the new mail recommendation is made. On the other hand, users who have already configured their mail in accordance with **MH** should be able to continue using **exmh** or **mh** as long as it works, albeit without the advantages of a distributed mail system.

New users may want to choose **pine** instead, since its native structure does support IMAP. The disadvantage to **pine** is, of course, its non-GUI, single-window user interface. If you plan to move over to an IMAP-based mail handler when it becomes available, and you choose **pine** in the interim, we recommend that you *not* configure **pine** to store messages in **MH** format.

Since **pine** is well documented elsewhere, this chapter provides only some start-up information in section 12.2.1 along with pointers to further information.

We cover the UNIX **MH** (Message Handling) products including both the standard line mode mail reader (referred to as **mh**), and a GUI version of **MH** called **exmh**. And we provide the information needed to get you up and running in **mh** or **exmh** as quickly and easily as possible, with a minimum of up-front configuration.

Appendix F contains information on how to customize **exmh** and **mh**, and how to take advantage of some of the less essential but very useful features. If you wish to do more in-depth customization of the **MH** products, we recommend the excellent reference book *MH and xmh - Email for Users & Programmers* by Jerry Peek, published by O'Reilly & Associates. An HTML version of this book can be accessed through the *UNIX Resources* Web page.

Appendix G provides a quick reference for the **mh** commands described in this chapter.

Appendix H contains instructions on moving your VMS mail folders to UNIX and converting them to the **MH** system format and organization.

You'll also find a brief discussion of **Berkeley mail** at the end of the chapter, only because it is a mail reader available on all UNIX machines.

12.1 Mail Forwarding

12.1.1 The Fermilab Mail Server: FNAL

First, a general word on mail handling at Fermilab. FNAL formerly referred to the central Fermilab VAX computing cluster, where most people at the lab had an account. As computing became more distributed, a single mail serving node was set up to handle the mail flow. This mail server was designated as FNAL, and the central VAX cluster was renamed to FNALV. FNAL will continue to function as the Fermilab mail server.

The central mail server provides a simple, uniform mail address for all users at the lab:

***username*@fnal.gov**

We strongly recommend that you take advantage of this capability and have all your mail routed through FNAL. It is your responsibility to choose a node for reading mail and to make sure your forwarding address on FNAL is set to this node. We step you through the forwarding process in section 12.1.3. However, we recommend that you work your way through the material in this chapter in the order it is presented, and change your forwarding address on FNAL only when you're ready to actually switch your mail activities to your chosen mail system on UNIX.

12.1.2 Forwarding on File-Sharing UNIX "Clusters"

Before getting started, you will need to choose a node for your mail activities. Once you do, forwarding mail to that particular node requires creating a simple `.forward` file in your `$HOME` directory. We show you how to do this in section 12.3 and again in F.1. If your chosen mail node is part of a UNIX "cluster" like FNALU on which a file-sharing system is installed (for example AFS or NFS), then creating a single `.forward` file in your common `$HOME` directory is sufficient for the entire cluster. The contents of this file indicates your chosen mail node within the cluster.

You might wonder why you need to choose a particular node in the case of a file-sharing cluster. The reason is that incoming mail on UNIX is sent to a non-shared area of *one* node; you can't change that, you can only specify *which* node you want used. From here the mail needs to be incorporated into the mail system of your choice; we cover this topic in sections 12.3.3, 12.4.3, and F.4. If you are working on a different node from the one receiving the incoming mail, you will be able to send mail, but you will only see previously incorporated mail; you will not receive (and cannot incorporate) new mail because it is inaccessible to this node.

12.1.3 Recommended Forwarding Procedure

Advantages

The procedure we describe here for setting your mail forwarding has three advantages:

- Every time you change your mail node, you only need to change the forwarding address on three nodes: FNAL, the old (or current) mail node, and the new mail node. If you have accounts on many systems, this is a timesaver.
- All Fermilab users have a standard, stable email address: `{username}@fnal.gov`.
- No mail is left sitting in the system mailbox (except as yet unincorporated mail on your chosen mail node).

Set Forwarding on your New Mail Node

As a first step, in your `$HOME` directory on your chosen UNIX mail node, create a `.forward` file using your preferred editor. Enter the following expression as the contents of this file:

```
\{username}@{node}.fnal.gov
```

where `{username}` is your login id on your chosen mail node, `{node}`. For example, user "fred" who wants to receive mail on the FSUI01 node of the FNALU cluster would enter the line:

```
\fred@fsui01.fnal.gov
```

This expression is an instruction to put incoming messages in this designated node's spool area. The initial backslash (`\`) is required to prevent looping.

Later, you may decide to set up mail notification and/or unattended autoincorporation on your chosen mail node. These options are discussed in Appendix F. If so, you will need to replace the above expression with a different one in the `.forward` file on this node (or cluster) in order to prevent the redirection of incoming messages to two or more places.

Set Forwarding on FNAL

As a second step, you need to log on to the mailserver node FNAL and use the menu-driven program to change the forwarding to your new mail node. If you don't know or don't remember your FNAL password, send mail to *compdiv@fnal* and request a new one.

Make sure you set forwarding here on FNAL to your new mail node BEFORE you change the forwarding on your old mail node to FNAL! Otherwise you get an infinite mail loop.

Once you're logged on to FNAL, you'll see the following menu:

```
                              VMS Menu System V3.0
    User: USERNAME
    Date: 11/10/95
    Forwarding Address:   username@FNALV.fnal.gov

    Fermilab Mail Server Menu
        F  Set Forwarding
        P  Set Password
        U  Unsubscribe from a LISTSERV
        X  Exit the Menu


    Time: 12:33 PM
    Select Option:
```

Select option `F Set Forwarding`. This brings you to the following screen:

```
Please enter your new forwarding address below

 Supported addressing formats

 DECnet     Node::User-ID                          FNALV::JONES
 BITNET     User-ID@Node                           JONES@FNALV
 INTERNET   User-ID@Node.domain                    JONES@FNALV.FNAL.GOV
 Quickmail  User-ID.Mailcenter@QMGATE.FNAL.GOV     JONES.Mailcenter@QMGATE.FNAL.GOV


 Please do not enter quotes or standard prefixes (smtp%, JNET%, IN%...)


 Your current forwarding address is:     username@FNALV.FNAL.GOV


 Enter forwarding address [ ? for help]:
```

Set your new forwarding address here in the format:

`{username}@{node}.fnal.gov`

where `{username}` is your login id on your chosen mail node, `{node}`.

Set Forwarding on all Other Nodes (including old mail node)

On all other external systems where you might receive mail (excluding the cluster containing your new mail node), UNIX or otherwise, set your forwarding address to your FNAL address.

To do this on other UNIX systems, create a `.forward` file in your `$HOME` directory on each separate node or file-sharing cluster. The contents of this file should be a single line containing the expression:

`{username}@fnal.gov`

where `{username}` is your login id on FNAL, the mail server. Note that there is no initial backslash here.

For information on how to set forwarding on non-UNIX machines, consult your system-specific documentation.

Set your "Reply To" Address

In most mail handlers, when a user directly "replies" to a message that you have sent (as opposed to starting a new message in the normal fashion to send to you), the reply is sent by default to the address from which you sent the original message, i.e., to the address in the "from" field. At Fermilab, this is not desirable; we prefer that all messages, including replies, get addressed to `{username}@fnal.gov`. In order to accomplish this, the outgoing message must specify a "reply to" address, and the mail handler that receives the message should be configured to send replies to that address in preference to the "from" address.

For **pine**, you need to set some configuration parameters. We describe this under **Configuring pine** in section 12.2.1.

For the **MH** mail handler, you can set your "reply to" address in the file `components`. We describe how to do this in section F.2.2. When replying to a message, the default behavior is to use the address in the "reply to" field if it has been supplied, otherwise to use the "from" address.

12.2 Overview of Mail Systems Available at Fermilab

As is typical for UNIX, there are many mail handling systems from which to choose. Here we provide a brief introduction to the currently available mail handlers at Fermilab. The Computing Division currently provides both **pine** and the **MH** system (which includes **exmh** and **mh**). Please see the remarks at the beginning of this chapter regarding upcoming changes in available mail handlers.

The simpler, less functional **Berkeley mail** is a mail reader that you will find on all UNIX machines. Therefore we give a cursory treatment of it in this chapter, although we explicitly do not recommend it.

> First choose one UNIX node for your mail activities.
>
> Secondly, we advise you to become familiar with the different mail systems and select the one you want to use as your primary mail handler *before* you start your conversion process. Although **MH** and **pine** can be made to work together (see section 12.2.1), they were designed independently of each other, and by default organize and handle mail differently.

12.2.1 pine

pine (Program for Internet News and Email) is a popular menu-driven, non-graphic UNIX mail system which the Computing Division supports. It can be configured to be compatible with the **MH** message storage structure, although it does not come this way "out-of-the-box".

The Computing Division no longer recommends using the **MH** message storage structure for **pine**.

We have not documented **pine**'s functionality in this manual; however we provide some start-up information for you. You will need to select from the wide variety of available documentation on the Web and elsewhere to learn how to use **pine**.

As a starting point for documentation on **pine**, see the *UNIX Resources* Web page. You'll find information under the **Mail** heading. In addition, also from the *UNIX Resources* page, select **Other UNIX Resources** followed by the *UNIX Reference Desk*, and click on *Applications* to find a good introductory document on **pine**.

Using pine and MH Interchangeably

This is no longer recommended; see the introductory remarks to this chapter.

If you have converted your mail to UNIX using the methods described in Appendices A and H, your mail folders are in **MH** format. See section 12.4.2 for more information on **MH**-style folders. If you plan to use **pine** and one of the **MH** interfaces interchangeably, you should leave your folders in **MH** format, and configure **pine** to recognize them. This involves creating appropriate mail folder collections. Follow these steps to make message storage in **pine** compatible with **MH**:

1) Run **`setup mh`** or **`setup exmh`** to create your **MH**-style `inbox` folder.

2) Run **`setup pine`**

3) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**. This creates the **pine** configuration file `$HOME/.pinerc`.

4) Exit **pine** (type **`q`** to quit).

5) Make sure the product **mailtools** is setup, then run the command **`fmh2pine`**[1]. This modifies the **pine** configuration, setting your **pine** inbox path to the default **MH** `inbox` folder, and adding an **MH** folder-collection containing all your **MH** folders. This command also makes your nested **MH** folders available to **pine**.

6) Run **`pine`** again.

Type L to select a folder to view. When you expand the folder collection lists and view the folders, you will find that the contents of your **MH** `inbox` folder is included in `INBOX` in **pine**'s `<mail/[]>` folder collection. Your other **MH** folders show up in the `<#mh/[]>` folder collection with their original names. Any nested **MH** folders you have defined will show up in separate folder collections according to their "parent" folders.

Anytime you add or delete an **MH** folder, run **`fmh2pine`** to update **pine**'s configuration accordingly.

If you normally use **exmh** and only use **pine** from home, for example, include the commands **`setup mh`** and **`inc`** in your `.login` or `.profile` file so that new mail gets incorporated and is displayed in **pine**.

Configuring pine

We recommend that you configure a few additional options.

`fmh2pine` sets `folder-collections` for you to the two values shown below thus configuring **pine** for use with **MH**. (This is no longer recommended.)

In **pine**, messages are not by default moved from the system spool area on your mail node (where they are initially deposited) into your home area. See **The UNIX System Mailbox** under section 12.4.3 for a discussion of the spool area. We strongly recommend that you properly set the options that move messages into your home area (the variables `default-fcc` and `read-message-folder`, and the on/off feature `auto-move-read-msgs`). There are several reasons for this, one of which is that if your mail node goes down, you can not access any messages in the spool area.

Go to **pine**'s SETUP CONFIGURATION menu. There are many options available, and they come with on-line help. With the cursor on the option you want, press **`?`** to get help on it. Some options are variables which take one or more values. To set a value for one of these variables, first type **`A`** to add, then enter the value. Or type **`C`** to change an existing value. Other options are in the form of a list of features that you can turn on or off individually. Enter **`X`** to enable a feature; leave unwanted features blank. Here we provide some recommendations:

`default-fcc` — The default folder for messages you send out; make sure it is *not* set to (uppercase) `INBOX`. Default is `sent-mail`.

`read-message-folder` — The folder in which read messages are placed. Make sure it is set, and that it is *not* set to (uppercase) `INBOX`.

`user-domain=fnal.gov` — This value specifies the domain part (right-hand side) of your return address on outgoing email, thus setting your "reply to" address to `{username}@fnal.gov`. This value is also used as the default domain for email that you send to a local username.

`reply-always-uses-reply-to` — Set this option so that **pine** uses the "reply to" field, if present, when you reply to incoming messages.

---

1. **`fmh2pine`** is part of **mailtools v2_2**.

`folder-collections=mail/[]`

This enables your basic set of **pine** mail folders.

`folder-collections=#mh/[]`

This enables your **MH** set of mail folders (no longer recommended; see introductory remarks for this chapter).

`default-fcc=#mh/outbox`

Here we are assuming that in **exmh** you send your copies of outgoing mail to the folder `outbox`. Set this to your default outgoing folder to make your copies of outgoing mail available to you when you get back to **exmh**.

The option list is rather lengthy, so we provide some further suggestions for you. We have found the following features to be particularly useful. Explanations for them are in the on-line help:

```
auto-move-read-msgs
auto-zoom-after-select
auto-unzoom-after-apply
customized-hdrs
default-composer-hdrs
enable-aggregate-command-set
enable-alternate-editor-cmd
enable-bounce-cmd
enable-flag-command
enable-full-headers-command
enable-jump-command
enable-suspend
enable-unix-pipe-cmd
enable-expanded-view-of-addressbooks
preserve-start-stop-characters
quell-user-lookup-in-passwd-file
```

Using pine Exclusively

If you choose to use **pine** exclusively, you will need your folders in the **pine** format. First, get ready to run **pine**:

1) Run **`setup pine`**

2) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**.

3) Complete the configuration as described in the above sections, however do not run **`fmh2pine`** and do not set `inbox-path` to `#mh/inbox`.

You can copy entire folders between the **pine** and **MH** collections as follows:

1) Go to the `FOLDER INDEX` screen for the desired folder.

2) Enter **`;`** (semi-colon) to issue the **pine** Select command.

3) Enter **`A`** to select All messages.

4) Enter **`A`** to Apply the command.

5) Enter **`S`** for Save.

Use <Ctrl-n> or <Ctrl-p> as necessary to choose the destination folder collection, then enter the new destination folder name. Or just type in a folder collection and name. Respond `Y` to create the new folder.

To use **pine** exclusively, you'll want to move any folders in the `<#mh/[]>` folder collection to the `<mail/[]>` folder collection.

Printing from pine

You will find it useful to set up printing from **pine**.

1) Go to **pine**'s SETUP PRINTER screen (enter `S` followed by `P`).

2) Move down to the `Personally selected print command` option. Enter `S` to select this option.

3) Enter `A` to add a printer, then as prompted, enter the name of the printer (this name is for your reference only) and the full command for the printer. For example:

   Printer: ***printer_name***
   Command: `flpr -q`***printer_name***

4) You may enter several printers. Enter `S` to select the printer you want to use.

5) Exit (`E`) and save changes.

12.2.2 MH Graphical Interface: exmh

exmh is the GUI implementation of **MH**, requiring the X environment. It is built with **Tcl/Tk**. It is quite customizable, yet for the most part satisfactory with a minimum of configuration. **exmh** provides interfaces to many other powerful tools including a spell-checker, a fast-search mechanism, **www**, and more. **exmh** can read and send MIME (Multipurpose Internet Mail Extensions) multimedia mail.[1] Note that **exmh** displays nonprinting characters as their ASCII hex value. For example, embedded line-feeds (often included in mail messages originating from a PC) are shown as `\x0d`.

Although we used to recommend **exmh** for users who most often work in the X environment, we no longer recommend that new users choose it, due to its uncertain future.

12.2.3 MH Line-Mode Interface: mh

mh is the command line implementation of the **MH** system, and as such is suitable for occasional use (when X-capability is unavailable) by those whose primary mail handler is **exmh**. **pine** may also be used in this capacity (see section 12.2.1).

mh is a very rich, flexible, and configurable mail system on its own. **MH** commands are not a part of a monolithic mail environment; each command is a separate program and the commands are issued directly from a UNIX shell prompt. Therefore, all the features of UNIX shells (pipes, re-direction of I/O, and so on) work with **mh**. You can use the man pages to get information on the **MH** commands.

12.2.4 Berkeley Mail

Berkeley Mail is an older mail system, and it is available on all UNIX systems. We do not recommend using it, but like the **vi** editor, knowing a little about it may come in handy sometimes! See section 12.5 for information on basic commands.

---

1. MIME allows you to read and send image files, sound files, and other items that require encoding to be sent via the mail system. See section 9.6. The O'Reilly book referenced at the beginning of the chapter includes an introduction to MIME.

12.3 The exmh and mh Mail Handlers

In this section we provide a guide to the basic commands and operations in both **exmh** and **mh**. Be aware that many more features and functions are available in these products than you'll find documented here.

We want to remind you that both **mh** and **exmh** are highly configurable interfaces to the powerful **MH** system. You can customize your personal setup to your heart's content, however we emphasize that neither **mh** nor **exmh** *requires* tweaking and tailoring beyond a very manageable minimal amount. We provide you with the information necessary to set up these interfaces for effective use in three places:

- We show you how to change fonts and editors in the basic command guide that follows in this section.
- In section 12.4 you'll find information on signature lines, mail aliases, and customizing the **exmh** folder display. We also provide information here on using **exmh** functions to incorporate mail.
- In Appendix F we discuss several nonessential but commonly tailored items, including customized mail notification (F.1.2) and message headers (F.2).

Recall that **exmh** and **mh** use the same underlying commands, although they have very different user interfaces. We've organized this section by function, showing you how to perform each selected function in both **exmh** and **mh**. This should make it easier to switch back and forth between the two interfaces if you sometimes find yourself at a graphics-capable terminal, and sometimes at an ASCII-only terminal.

We recommend that you perform couple of initial steps before starting to practice. These steps relate to mail forwarding, and we recommend them at this stage because of the way incoming mail is stored under UNIX (mail storage is discussed in section 12.4.3).

- Choose your mail node, and plan to do your practicing there.
- Create a file named `.forward` in your `$HOME` directory on this node containing the single line:

```
\{username}@{node}.fnal.gov
```

where `{username}` is your login id on your chosen mail node, `{node}`. This ensures that your "practice mail" will be available to you on the right node if you should need it later.

If you have an account on another machine, why don't you send a message or two to yourself on your chosen mail node before getting started so you have some messages to read? This isn't necessary, you can always send messages to yourself on your mail node.

12.3.1 Run Setup and Invoke the Application

exmh

```
% setup exmh
```

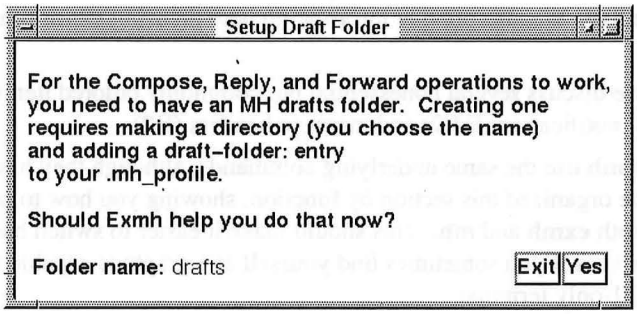
The use of **exmh** requires that you set your *DISPLAY* environment variable appropriately (see section 9.2).

When you setup **exmh**, you also setup several other products[1] which **exmh** either needs or with which it can interface. Among these is **mh**. If you've never run **`setup mh`**, then the first time you run **`setup exmh`** the file `.mh_profile` is created in your `$HOME` directory, and a message is sent to you (ignore the content of this message; it may be out of date). The `.mh_profile` file defines your basic **MH** configuration.

To invoke **exmh**, enter:

```
% exmh [&]
```

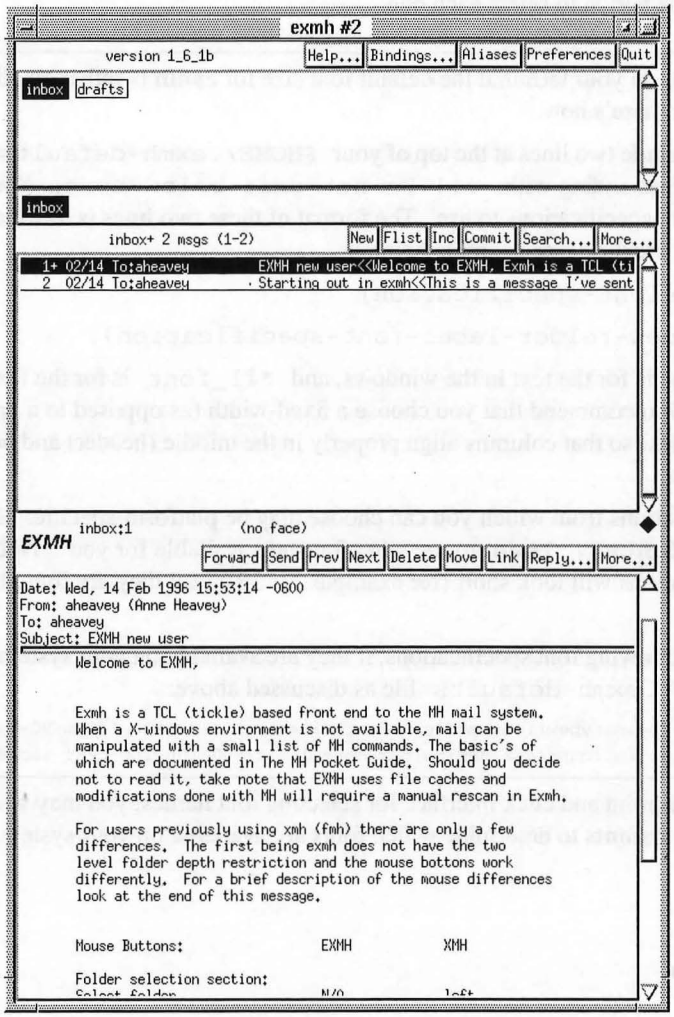
The first time you invoke **exmh**, you are asked if you want to create a message folder for message drafts in your `Mail` subdirectory. If you respond Yes, it adds a line to your `.mh_profile` identifying the folder.[2] By default the folder is called `drafts`, but you are given the opportunity to name it as you wish.



Click on Yes if you wish to continue.

1. Run the command **`ups list -l exmh`** to see the list of product dependencies.
2. It will also add the lines `unseen-sequence: unseen` and `repl:` to your `.mh_profile`.

Now an **exmh** window appears. The **exmh** window is split into three areas, which we also call windows. The title bars in each of the three windows have command or pull-down menu buttons. Actions are performed by clicking on the buttons and/or choosing from menu items.



The three windows have different functions:

Top — displays one "button" for every defined message folder. The current folder button is highlighted. Select a folder (make it current) by clicking on it with the left mouse button. **MH** message folders are discussed in section 12.4.2.

Middle — displays message headers in the current folder (the folder selected in the top box). The title bar for this area displays the current folder name and the number of messages. The header of the current message has a '+' next to its number and is highlighted.

Bottom — displays the current message (the message highlighted in the middle box). The title bar for this area displays the current folder name and the current message number.

A general usage note: You can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. To select multiple headers (useful for refiling, removing, etc. several messages at a time), hold down the shift key while you press the left mouse button to select each one.

You may find that on your terminal the default font size for **exmh** is rather small. If you really want to change it, here's how.

You'll need to include two lines at the top of your `$HOME/.exmh-defaults` file (above the three comment lines ending with: `!!! Do not edit below here`). You will need to choose which font specifications to use. The format of these two lines is as follows (don't include the curly brackets, see example below):

```
*font: {new-font-specification}
*fl_font: {new-folder-label-font-specification}
```

The `*font` line is for the text in the windows, and `*fl_font` is for the text in the folder label buttons. We recommend that you choose a fixed-width (as opposed to a proportionally spaced) font for text so that columns align properly in the middle (header) and bottom (message display) windows.

The font specifications from which you can choose may be platform-specific. Execute the command **`xlsfonts | less`** to see what fonts are available for you[a]. Note that some of the font specifications will look short (for example `vt33014`); they are font aliases, and will work here, too.

We suggest the following font specifications, if they are available on your system. Just add these two lines to your `.exmh-defaults` file as discussed above:

```
*font: -b&h-lucidatypewriter-medium-r-normal-sans-14-100-100-100-m-80-iso8859-1
*fl_font: -b&h-lucidabright-demibold-r-normal--14-100-100-100-p-84-iso8859-1
```

a. **xfontsel** is a point and click interface for selecting font names; you may want to use this in addition to **xlsfonts** to determine which fonts are available on your system.

mh

To setup **mh**, enter:

```
% setup mh
```

You don't need to do this if you've already setup **exmh**.

The first time you do this, it creates a file `.mh_profile` in your `$HOME` directory. It also automatically sends a message to you (ignore the content of this message; it may be out of date).

Now all the **mh** commands are available for use. As mentioned earlier, there is no single invoking command; you simply type **mh** commands at the shell prompt as you need them.

12.3.2 Compose and Send Messages

Remember to send yourself a message or two so you'll have something to read later on.

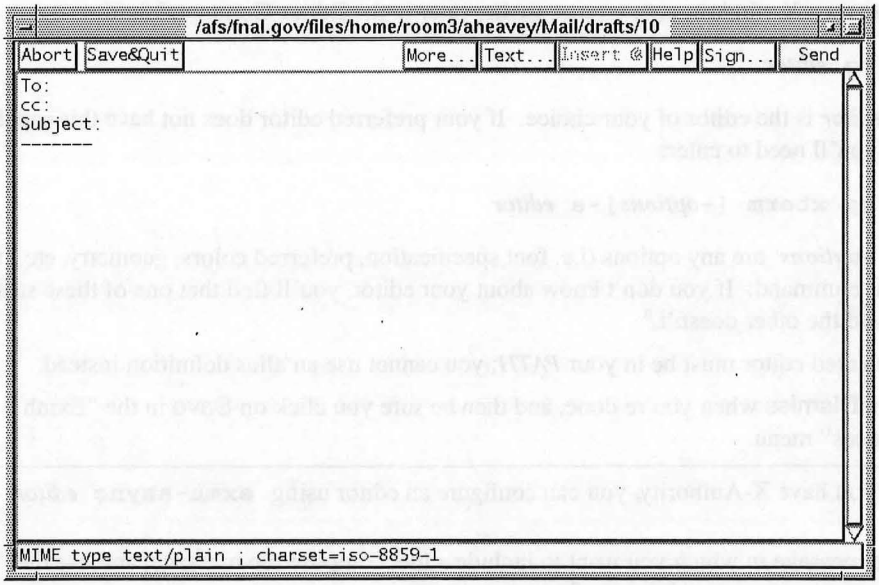
For both **exmh** and **mh**, a default message header with the following contents is provided for sending messages:

```
To:
cc:
Subject:
--------
```

If you wish to change the contents or appearance of the message header, you can create your own `components` file. See section F.2 for information.

exmh

To compose your message click on send in the bottom window. The word *comp* (for compose) will appear in the command menu bar. Then a new drafts window will pop up in which you will write your message. **exmh** uses **sedit** (Simple Editor) as the default editor for composing messages. As a new user, you may just want to use this editor for a while to simplify your initial setup.



Fill in the information, using a tab between the header fields[1], and then type in the body of your message below the dashes. On many terminals <Ctrl-h> is used for character deletion. Alternately, you can select the text to delete and press your Backspace or Delete key[2]. Most mail programs expect a carriage return at the end of each line of the message, so it is always wise to include it.

---

1. Note, however, that if you tab twice in rapid succession, the cursor will move down to the message content area. You can use this feature to pass over all remaining fields if you don't want to enter information in them.
2. If the backspace key doesn't work properly when sending mail, you can rebind it by first clicking on Bindings... in the top line, then click on the Simple Edit submenu. In the backspace field, type **`Key-Delete`**. Click on Save.

Note that if you set your preferences (go to Preferences button in top window) for Simple Editor to Format Mail Default *On*, you don't need to explicitly include the carriage return at the end of each line, it will be done automatically for you after you exit the editor. This feature is only available for the **sedit** editor.[1] Once you finish in the "Simple Editor Preferences" window, click Dismiss. Then click Save on the "Exmh Preferences" window.

Once you have completed the message, you can send it to the recipient(s) by clicking on the send button. If you decide not to send it, you can either save it in the `Mail/drafts` folder by clicking on Save&Quit, or discard the composition by choosing Abort. There are many useful options available on the drafts window under More... such as Insert File and Spell. Under Text... you have some options for fonts, which only are useful if the destination machine/mail reader supports them. If not, the recipient will see things like `<bold>{your message text}</bold>` in the message.

If you don't want to use this editor, you can define a different default editor, or you can define a secondary editor and then select it from the More menu in the compose window (this secondary editor option is only available with **sedit** as primary editor).

To define a different default editor, select Editor Support under the Preferences menu option in the top window. A pop-up window will appear. If your preferred editor "knows how to" create its own X window, e.g., **emacs**, **nedit**, enter in the Editor Command field:

`exmh-bg` ***editor***

where ***editor*** is the editor of your choice. If your preferred editor does not have this capability, e.g. **vi**, you'll need to enter:

`exmh-bg xterm [-`*options*`]-e` ***editor***

where **`-`*options*** are any options (i.e. font specification, preferred colors, geometry, etc.) to the **`xterm`** command. If you don't know about your editor, you'll find that one of these strings works and the other doesn't.[a]

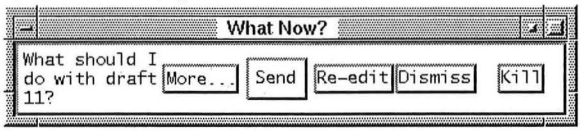
Your selected editor must be in your *PATH*; you cannot use an alias definition instead.

Click on Dismiss when you're done, and then be sure you click on Save in the "Exmh Preferences" menu.

a. If you have X-Authority, you can configure an editor using **`exmh-async`** ***editor***.

To send a message in which you want to include a file, compose the message, and use the "include file" command appropriate for the editor you're using.

When you save the composed message you are presented with a "What Now?" dialog box (a feature of **MH**) that gives you options to Send, Re-edit, Dismiss (this "dismisses" the pop-up window and leaves the message in the drafts folder), or Kill the message.



1. However, be aware that the line breaks inserted by **sedit** after you exit the editor may not be the places that *looked* like line breaks while you were in the editor. The line wrapping width is controlled by the value of Max Line Length in the Simple Editor Preferences window, not by the width of the editing window.

mh

mh will throw you into **vi** by default for composing a message. This is also true for forwarding and replying to messages. You can specify a different editor on the command line if you like.

To compose a message in **vi**, enter:

`% comp`	This brings up a **vi** session. Compose your message. Then save the message file, and exit the editor.

Remember to have the handy list of **vi** "survival commands" from section 11.3.1 with you! Just in case you get stuck, here's your emergency exit: hit the Escape key (or <Ctrl-[>) and then enter:

`:x`	to save and exit, or
`:q!`	to quit without saving

Once you exit the editor, **MH**'s **What Now?** program is automatically invoked to prompt you for an action. At the `What Now?` prompt, you can enter a carriage return to get a list of options. Normally you will enter one of these options:

`send`	to send the message
`quit`	to save the message in `Mail/drafts` but not send it
`quit -delete`	to quit and not save.

To compose a message in a different editor, enter:

`% comp -e` ***editor***	This brings up a session in your selected editor. This option will override any editor defaults. Compose your message. Then save the file, exit the editor, and respond to `What Now?`

If you *really* want to avoid **vi** at all costs, we'll show you how to set a different default editor.

> To change your default editor for **mh**, add the following line to your `.mh_profile` file:
>
> `Editor: {editorname}`
>
> where `{editorname}` is your preferred editor, for example:
>
> `Editor: emacs`

If you want to send a text file rather than a message, use the `mhmail` command. Here is an example to illustrate the syntax (more options are available, see the man page):

```
% mhmail joe@fnal -cc sue@fnal -subject "Here's the file!" < \
  filename
```

To send a message in which you want to include a file, compose the message, and use the "include file" command appropriate for the editor you're using. In **vi** enter:

`:r` ***filename***

In **emacs**, use [Ctrl-x] followed by `i`, then type in the filename at the prompt.

12.3.3 Incorporate and Read Incoming Messages

Mail incorporation refers to the process of moving incoming mail messages from the default mail holding area on your UNIX node into your personal area. Section 12.4.3 discusses this in more detail. You can't read messages until you incorporate them. As messages get incorporated according to **MH** default standards, they go to the directory `$HOME/Mail/inbox` (`inbox` is

called a folder), which was created for you during setup. Each message becomes a separate, numbered file in `inbox`. When you incorporate messages, you get all the messages in the spool; you cannot choose to incorporate some and not others.

exmh

There are a few ways to incorporate mail during an **exmh** session. You can incorporate your messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

To incorporate messages manually from within **exmh**, click inc from the middle window's menu. By default your spooled mail is placed in your `inbox` folder. You should now see the headers of the messages you sent yourself in the middle window. (If you didn't already send yourself some mail, go back and do it now.)

Once you incorporate your message(s), you can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. The message displays in the bottom window.

To set up immediate or periodic incorporation during your **exmh** session, see section 12.4.3.

mh

First, incorporate the new message(s) into the **MH** system using the command:

```
% inc
```

This will display the headers of the incorporated messages, with a plus sign (+) next to the current message. The current message is the first unread one, usually the first one you just incorporated unless you had unread messages already in the folder. (If you didn't already send yourself some mail, go back and do it now.)

After incorporating, you can use **`scan`** to display the message header(s):

`% scan`	Display message headers from the current folder (since we haven't created any other folders, the current one is `inbox`).
**`% scan last:`*`n`*	Display the last *n* message headers from the current folder.
`% scan` ***`m-n`***	Display message headers *m* through *n*, inclusive, from the current folder.

Here's our first example of using **mh** commands together with standard UNIX commands. To select message headers in `inbox` according to a ***pattern***, you could pipe **`scan`** to **`grep`** in this way (the **`+inbox`** specifically requests the `inbox` folder, and the **`-i`** option of **`grep`** indicates a case-insensitive search for ***pattern***):

```
% scan +inbox | grep -i pattern
```

To read the current message, enter:

`% show [`*`n`***`]`**	Display the current (or specified) message.

Note that the **`show`** command can be used with more than one message number. For example **`show`** *n* can be expanded to **`show`** ***m n ...*** or to **`show`** ***m-n***.

Other commands to read messages are:

`% next`	Display the next message.
`% prev`	Go back and display the previous message.

12.3.4 Reply to Messages

The default reply header looks like this:

```
To:
cc:
Subject:
In-reply-to:  Your message of "Day, date time"
              <message-id>
--------
```

If you don't like the standard reply header, you can reformat it by creating your own `replcomps` file. See section F.2 for information.

When you want to include the original message in your reply, it is often convenient to precede each line of the original message text by a `>` symbol, to distinguish it from your text. FUE sets this up automatically for **exmh** and **mh**.

This formatting can be inconvenient if the recipient needs to work with the original text, and thus needs to remove all the `>`'s. You can avoid this by *forwarding* the message back to the original sender, instead of replying. Forwarding is covered in section 12.3.5.

If you want to change the format of your replies, you can create a file called `.mh_filter` in your `$HOME` directory. The O'Reilly **MH** book describes the different possible ways of setting this up. The contents of the default `.mh_filter` is:

```
body:component="> ",compwidth=0
```

exmh

Select the message to which you want to reply so that it displays in the message (bottom) window. Then click on the Reply button in the bottom window. You get a menu of options. Reply to sender sends it just to the sender, while Reply all sends it to everyone listed in `To:` and `Cc:`. To include the original message in the reply, choose one of the Reply menu options with /include.

Once you have made a selection from the Reply menu options, a composition window pops up. The header fields will be filled in. You can edit the header if needed. Move the cursor to the body area and type in your reply.

For **sedit**, click on the send button to send the reply. To skip the reply, click on Abort.

For other editors, save the message and exit the editor. Then respond to the "What Now?" pop-up menu.

mh

By default, the **`repl`** command uses the **vi** editor. To avoid this, either use the **`-e`** ***editor*** option or set a different default editor in your `.mh_profile`.

To reply to a message, enter:

`% repl [`*`n`*`]`	Reply to the current (or specified) message. By default, everyone who got the original message gets a copy (the sender, plus everyone in `To:` and `cc:`).
`% repl_inc [`*`n`*`]`	Like **`repl`**, but include the original message. (This FUE command replaces the standard **mh** command **`repl -filter`** ***filter_file***.)

Either of these opens an editing session for the reply. As with **`comp`**, save your message and respond to the `What Now?` prompt.

The **-query** option used with either of these commands prompts about each recipient separately. To reply to selected addresses in the `To:` and `cc:` fields of the original message, enter:

% repl -query Reply to the current message. You will be prompted for each address in turn, to which you reply `y` or `n`, for example:

```
Reply to user1@FNAL.GOV? y
Reply to user2@xxx.yyy? n
```

Use the **-nocc** ***headerfield*** option with either command to prevent all addresses in a particular header field from receiving the reply; for example:

% repl -nocc cc ***[n]***

Reply to the current (or specified) message, including all addresses except those in the original `cc` list.

12.3.5 Forward Messages

By default, forwarding uses a header file called `forwcomps` in the **MH** library directory. This file is similar to that used for the compose function (`components`). If you want your header for forwarding messages to be identical to that for sending messages, you can use the components file for both. Just include the following line in your `.mh_profile` file:

```
forw: -form components
```

If you want to customize the header for forwarding mail, you need to set up your own file called `forwcomps` in your `$HOME/Mail` directory. Edit it in the same manner as `components` (see section F.2).

exmh

To forward a copy of a message to someone, first select the message. You can select more than one message at a time. Then select Forward from the bottom window. A composition window comes up with a blank header and the message text. Fill in the header, and edit the message if you wish. Send or abort as explained earlier, according to your editor.

mh

By default, the **forw** command uses the **vi** editor. To avoid this, either use the **-e** ***editor*** option or set a different default editor in your `.mh_profile`.

To forward a message, enter:

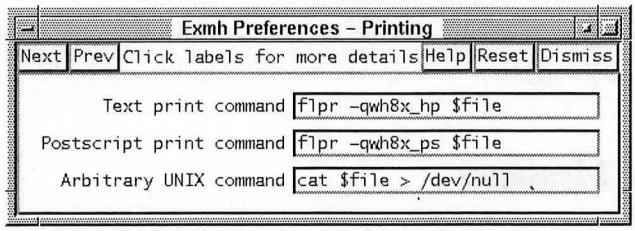
% forw ***[n]*** Forward the current message (or the specified message).

This opens an edit session. Fill in the header, and edit the message as you like. As with **comp** and **repl**, save your message and respond to the `What Now?` prompt.

12.3.6 Print Messages

exmh

First set your print queue by selecting Printing under Preferences in the top window. It should look like the following, with your own print queue substituted:



Click Dismiss when you've completed the change on the "Printing Preferences" window, then click Save on the "Exmh Preferences" window.

To print the current message, use the Print option from the More... menu in the bottom window.

To print several messages from a single folder, select them in the middle window, and use the Print option from the More... menu in the bottom window.

mh

To print a mail message, you use an **mh** display command piped to a UNIX print command (normally **`flpr`** under FUE). Any printing defaults you've set up for printing files will be used here (printing is covered in Chapter 8).

To print message number ***n*** on ***queue***, enter:

`% show` ***n*** **`| flpr -q`** ***queue***

The option **`-showproc`** with the **`mhl`** argument formats the message a little differently so that it's easier to read (and nicer for printing), and pipes it to **more**. You would type:

`% show -showproc mhl` ***n*** **`| flpr -q`** ***queue***

Especially good for printing multiple messages is the argument **`pr`** which separates mail messages onto successive pages:

`% show -showproc pr` ***m n ...*** **`| flpr -q`** ***queue***

12.3.7 Extract Messages

In **MH** format, each message is already a separate file, and can be manipulated as any other file. You may find however, that extracting messages is easier or quicker than standard UNIX file manipulation for some tasks.

exmh

To extract the current message to a file, use the Save to file option from the More... menu in the bottom window. A window pops up to allow you to choose the file to create or to which you want to append the message.

mh

To extract a message into a file use standard UNIX output redirection:

```
% show n > filename
```

This is essentially equivalent to the command:

```
% cp $HOME/Mail/folder/n filename
```

The show command includes one extra line that the **cp** command doesn't create:

```
(Message {folder}:{n})
```

12.3.8 Remove Messages

First, a note about message removal: When you use **rmm** in **mh**, or the delete button in **exmh**, the files are not actually deleted. They are renamed with a comma (,) or pound sign (#) in front so that **exmh/mh** doesn't recognize them as messages. The files are still there consuming disk space, and they don't go away unless YOU do something about them.

You can choose to periodically go through and clean out the old files that you no longer need in order to recover disk space. In **exmh**, use the Purge All Folders button (under the More menu in the middle window). This will delete the message files older than (default of) 7 days which you had previously "renamed". Or manually use the UNIX command **rm** on all files beginning with a comma (,) or pound sign (#).

A better solution is to add an entry to your `.mh_profile` changing the behavior of the Delete button /**rmm** command:

```
rmmproc: /bin/rm
```

This will really delete the message files when you use **rmm** (**mh**) or when you hit the Commit button after deleting messages (**exmh**). For clarity in the following text, we simply refer to deleting messages.

exmh

To delete one or more messages, first mark them for deletion by selecting the headers in the middle window then click on Delete from the bottom window. The message headers you selected will be highlighted; on some terminals they will be a different color or have a background pattern. If you are sure that you want to delete the marked messages, click on the Commit button in the middle window. If you change your mind before you commit, select the messages you do not want to delete and unmark them (UnMark is an option under More... in the bottom window).

Note that after selecting a message, the cursor may move down to the next one, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

Use the **rmm** command to remove messages:

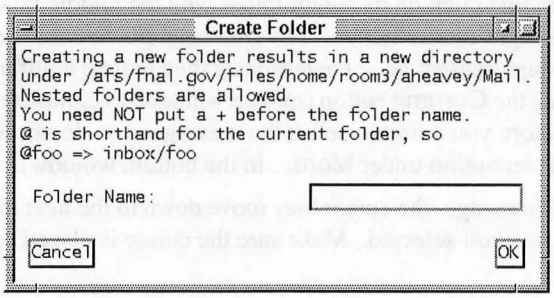
```
% rmm [n m ...]
```

Delete the current (or specified) message(s).

12.3.9 Create, Change and Remove Folders

exmh

To create a new folder from within **exmh**, select New from the middle window. The following window pops up for you to fill in. When you're done, click OK to create your new folder.



To switch current folders, click on the desired folder name in the top window with the left mouse button and the message headers from that folder will display in the middle window. It is now the current folder. You can read and work with the messages from this folder.

To view nested folders and their contents (see section 12.4.2), click the "parent" folder with the middle mouse button, then select a nested folder with the left mouse button.

To remove a folder, select it, then click on Delete folder under More... in the middle window.

mh

When you refer to a folder on the command line, precede it with a plus sign (+). To create a new folder, enter:

```
% folder +new_folder_name
```

There are a few ways to reset the current folder to a different one. The most explicit command for this is:

```
% folder +new_current_folder
```

Or, just running one of several commands with a different folder name changes the current folder until you again change it to something else. For example:

% scan ***+folder_name***	Display message headers from ***folder_name*** and set this folder to current.
% show ***+folder_name***	Display the current message in ***folder_name*** and set this folder to current.

Note that the **refile** command (see section 12.3.10) accepts folder name arguments but doesn't change the current folder.

To remove a folder, enter:

% rmf ***+folder***	Delete the specified folder.

12.3.10 Refile Messages

exmh

To refile one or more messages into a different folder, first select the header(s) in the middle window, then click the right mouse button on the destination folder in the top window. If they're moving to a nested folder, first click on its parent folder with the middle mouse button, and then on the nested folder with the right mouse button. The messages are now marked for refile. The message headers will be highlighted; on some terminals they will be a different color or have a background pattern. Click the Commit button (middle window) to complete the refile operation. If you change your mind before you commit, select the messages you do not want to refile and unmark them (UnMark is an option under More... in the bottom window).

Note that after selecting a message, the cursor may move down to the next message header, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

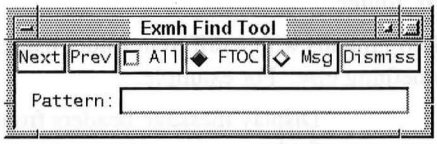
To move the current message (or specified messages) from the current folder to a different folder, enter:

```
% refile +folder [n m ...]
```

12.3.11 Search for Messages

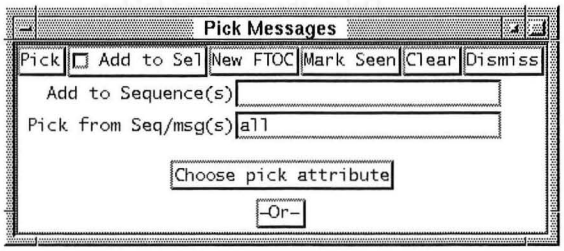
exmh

To search messages in a folder, first select the folder in the top window. Then select Search... from the middle window's menu and select the relevant option. You can get help on various items by looking into the search help. If you choose Find in table of contents, for example, you get a Find Tool window where you can type in a search pattern.



The headers of the "found" messages are highlighted. You can then display only the headers of these messages by choosing List only selected messages from the More... menu.

The Pick by Attributes from the Search... menu provides an interface to the MH `pick` command. This is a more sophisticated feature that allows you to set up complex logical search expressions. If you use the "before" or "after" search criteria, the date format accepted is `nn-mmm-yyyy`, for example `12-jan-1996`.



You have the option of saving the *sequence* of the picked messages. **MH** sequences are stored lists of message numbers. When you've finished setting up your search criteria, click Pick.

Another option you have is to use the **glimpse** utility, also available under the Search menu for full text searches. This requires building an index (use the menu option index under Glimpse full text), which may take a long time. For more information see the man pages for **glimpse**.

mh

`Pick` searches messages within a folder for the specified contents, and then identifies those messages. Two types of search primitives are available: pattern matching and date constraint operations. See the man pages for more information on `pick`.

`% pick -help` Display a list of pick options.

`% pick` ***options*** Select messages by content, date, sender, recipient, etc.

You can use `pick` to find messages sent from someone, sent to someone, on a given date, before or after a date, and even search every line of every message in the current folder. To tell `pick` to scan through every line of every message in the folder `recent_work` and return the message numbers with the word "quark" in them, enter:

```
% pick -search quark +recent_work
```

Here is an example showing the selection of messages within a date range:

```
% pick -after 1-dec-1995 -and -before 1-jan 1996
```

You can use `pick` with sequences, which are described above in the **exmh** portion. Commands that accept message numbers accept sequences as well; just type the sequence name. For example, find all messages received from ***sendername***, and store them in the sequence `picked`:

```
% pick -from sendername -sequence picked
```

The result also tells you how many messages are picked. You can then scan these messages by typing:

```
% scan picked
```

You can search for a string within the sequence by typing, for example:

```
% pick -subject 'quark substructure' -sequence picked
```

You can use `pick` and `scan` together (note the use of backquotes here):

```
% scan `pick -from sendername`
```

12.4 Basic Configuration for MH

12.4.1 Configuration Files

In order to take advantage of the variety of features provided by the **MH** package, several configuration files can be used for both **exmh** and **mh**. In this section we list all the files discussed in this chapter and in Appendix F, and provide brief descriptions for reference.

You're already familiar with two of them, `.mh_profile` and `.forward`.[1] See sections F.2.1 and F.1, respectively, for more information on these two files.

---

1. The `.forward` file is not specific to the **MH** system; it may be used with any mail handler.

`.mh_profile` defines your basic configuration (where mail is stored, the editor to use, various MH options, etc.). Running **`setup mh`** or **`setup exmh`** the first time automatically creates a default `.mh_profile` for you in your `$HOME` directory.

`.forward` stores your forwarding address. It is used in mail forwarding (routing), notification, and unattended autoincorporation. You must create it in your `$HOME` directory.[1]

The following three files allow you to customize your mail headers. See sections F.2.2, F.2.3, and F.2.4, respectively, for more information on them:

`components` defines message header format for sending mail. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`replcomps` defines message header format for replies. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`forwcomps` defines message header format for forwarded messages. If you don't create your own copy in your `Mail` directory or elect to use the `components` file header format, the system default file in the **MH** library section will be used.

In sections 12.3.4 and 12.4.5 we show you how to create:

`.mh_filter` allows you to include the original message in a reply. It goes in your `$HOME` directory.

`{alias file}` stores distribution lists and local aliases for long mail addresses (you define its name in your `.mh_profile`). It is assumed by default to be in your `Mail` subdirectory, but you can define it otherwise.

In Appendix F we provide information on yet a few more configuration files!

`scan-form` provides the format of display when you scan mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`inc-form` provides the format of display when you incorporate mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`.maildelivery` controls how local delivery is performed with manual or background incorporation in **exmh** when the presort option is selected. It also controls how delivery is performed when unattended automatic mail incorporation is defined in `.forward` using **slocal**. No default is provided; if you need this, you must create it in your `$HOME` directory.

12.4.2 MH Mail Folders

MH uses the concept of mail folders to store messages. Folders are simply subdirectories of your `Mail` directory which contain mail messages, one message to a file. To distinguish folders from files in **MH** commands, folders are prefixed with a plus sign (+) in a command.

---

1. If you only have an account on one machine (i.e. one node that is not part of a file-sharing cluster) besides FNAL and if you don't want automatic mail incorporation or notification, you don't need the `.forward` file.

For example:

```
% refile +admin
```

is an **MH** command that moves (refiles) the current message in the current folder to the folder `admin`.

Folders can be multi-level, for example `outbox/feb96`. Within a folder, the message filenames are incremented numbers starting at one (1). Running **`ls -l`** on a folder, you will see a numbered file for each message (in "alphabetical" order), for example:

```
total 296
-rw-r--r--   1 username  g020       339 Nov  9 11:36 1
-rw-r--r--   1 username  g020       864 Nov  9 11:36 10
-rw-r--r--   1 username  g020      3297 Nov  9 11:36 11
...
-rw-r--r--   1 username  g020       753 Nov  9 11:36 2
-rw-r--r--   1 username  g020      2628 Nov  9 11:36 20
```

The command **`ls -lt`** is convenient for listing them in reverse chronological order.

After a lot of refiling and deleting of messages, the messages in your folders may not be in order by date anymore, and you will undoubtedly have numerical gaps in the file names. To reorder and renumber the message files:

exmh	use the Sort folder and Pack folder options under the More... button in the middle window
mh	use the commands **`sortm`** and **`folder -pack`**, respectively

12.4.3 Incorporation of Incoming Mail into Folders

The UNIX System Mailbox

Due to the wide variety of mail readers on UNIX and the configurability provided by many of them, incoming mail is not by default delivered to any particular file or area in your home directory. Instead it is collected in a *system mailbox*, a directory defined as the mail holding area, on each node. In FUE, the environment variable *MAIL* is defined to reflect the system-specific location of your mailbox file, set to `/usr/spool/mail/{username}` or `/usr/mail/{username}`, depending on the UNIX flavor.[1] For example, user `fred` has his unread, unincorporated incoming mail stored in chronological order in the file `/usr/mail/fred`.

Your mail resides in this file until you incorporate it into a different file (mail folder). How and where it gets incorporated depends on the mail reader you decide to use. Here we discuss incorporation into the **MH** mail folder structure.

Options for Incorporating Mail

When new mail arrives, by default it resides the system mailbox until you incorporate it into your mail folder. There are a few ways to incorporate mail during an **exmh** session (i.e. "attended" incorporation), as we mentioned in section 12.3.3. You can incorporate messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

---

1. See the file `fermi.login` (for C shell family) or `fermi.profile` (Bourne shell family) in Appendix C. On a SunOS 5 (Solaris) system, you may have to set the variable *MAIL* manually due to a bug.

If you want to configure **exmh** so that it incorporates your mail at regular intervals and/or at particular times during your **exmh** session, first go to the Preferences menu in the top window.

- Choose the Incorporate Mail menu. Do you want to incorporate new mail when you invoke **exmh**? Check the box. Do you want to incorporate waiting mail when you click to open the **exmh** icon? Check the box. Dismiss this window and save your preferences.
- Choose the Background Processing menu. Do you want to incorporate new mail periodically? Check the Inc box. The period is set to 10 minutes by default, but you can change this value. Dismiss this window and save your preferences.

Would you like **exmh** to sort your mail for you, automatically putting some messages in a particular folder? If so, you need to do two things:

1) Create a `.maildelivery` file describing how you want your mail sorted (see section F.2.6).

2) Tell **exmh** to use this file. Go to the Preferences/Incorporate Mail menu, and under Ways to Inc, check the Presort box.

We discuss unattended automatic mail incorporation in section F.4. This is different from attended automatic incorporation in that mail is automatically incorporated whether or not you're running **exmh**, and whether or not you're even logged in. We do not recommend this method, although we include it here for completeness.

12.4.4 Signature Lines

To include a boilerplate signature to messages that you compose and send in **exmh** or **mh**, you need to create your own `$HOME/Mail/components` file, and simply add the signature to the bottom of this file, underneath the dashed line. For information on the `components` file, see section F.2.2. This file does not govern replying and may not govern forwarding (see section 12.3.5).

To add a boilerplate signature when replying to messages, you can add a signature to the end of your filter file, called `.mh_filter` in your `$HOME` directory (see section 12.3.4). This file gets used only when you're using the option to include the original text in the reply.

If you're using your own `$HOME/Mail/forwcomps` file for forwarding messages, include the signature lines at the end of this file. Note that your signature will appear *above* the text of the forwarded message.

12.4.5 Mail Aliases

You can send mail to many people at a time by defining an alias for the addresses of the recipients. You may also want to define aliases for long single addresses. Create a file (we call it `mh_alias` for example purposes; you can call it what you like) in your `$HOME/Mail` directory, and include an `Aliasfile` line like the following in your `.mh_profile`:

```
Aliasfile: mh_alias
```

This example alias file shows the required format:

```
d0spokes: mont@fnal.gov,\
          pgrannis@fnal.gov

cdfspokes: carithers@fnal.gov,\
       bellettini@fnal.gov
```

There cannot be any blanks after the backslash (\) in the aliases. To see your aliases you can use the command **ali**. For example if you type:

```
% ali d0spokes
```

you will get the response:

```
mont@fnal.gov, pgrannis@fnal.gov
```

12.4.6 Folder Order and Header Display (exmh)

Some further options available for **exmh** (but not for **mh**) to include in the `.mh_profile` file are:

```
Folder-Order: inbox folder2 folder3 * drafts
Header-Suppress: .*
Header-Display: Date To Cc From Subject
```

`Folder-Order` sets up the order of display of folders on your **exmh** window. Wildcards as used in filename expansion are allowed, so you only need to specify folders that you want displayed in particular places. In our example, we specify what folder to show first, second, third, and last. All the others will be sorted alphabetically between the third and the last.

`Header-Suppress` and `Header-Display` control which headers you see when you read a message. The example above indicates that everything is suppressed except the `Date:`, `To:`, `From:`, `Cc:`, and `Subject:` lines.

12.5 Berkeley Mail

In this section we discuss the simpler, less functional mail handler **Berkeley mail** because it is one that you will find on all UNIX machines. We describe how to read and send messages from the command line using the command **mail** or **fermimail**. In the Fermi files that run when you log in, the command **mail** is set to point to **Berkeley mail** by default (via the alias **fermimail**).

12.5.1 Send Messages and Files

To send a mail message, the command format is:

% mail [-s *subject*] *recipient* [< *filename*]

If you don't include the **-s** option with a ***subject*** (subject must be enclosed in quotes if it's more than one word), the system will prompt you for one after you hit carriage return. The ***recipient*** is a user name in standard format. If you want to send a file instead of composing a message, use the input redirection (<) with the ***filename*** (and path, as necessary). If you don't include a file, the cursor will be placed at the start of the first line under the command (no prompt) for you to start composing your message. Conclude the message by going to a new line and typing a period (**.**) followed by carriage return. You will be prompted for a `Cc:` before exiting. To abort a message that you are composing, use the standard *kill* control sequence (see section 2.4).

A very simple example command line using this format to compose a message is:

```
% mail butler@fnal
```

An example using the subject option and sending a file is:

```
% mail -s "Meeting minutes" butler@fnalb.fnal.gov < minutes
```

12.5.2 Read Messages

Assuming you have no autoincorporation set up, all of your unread incoming mail resides in the file designated by the variable *MAIL* (see section 12.4.3). When you log in, the system notifies you if this file contains any messages. It will not automatically notify you of incoming mail at other times during your session, unless you are in a mail reader.

To read your messages in the system mailbox, simply type:

```
% mail[1]
```

This puts you in command mode from which you can read messages. Enter `?` to get a list of commands. If there are messages, after you read them and when you are exiting mail, the system automatically sends the messages to your `$HOME/mbox` file, and deletes them from your system mailbox file. If there are no messages to read, the system will tell you so and exit.

For more information, see the man pages for **`mail`**.

1. Note that on some systems the command may be **`mailx`**, **`Mail`** or **`fermimail`**.

Chapter 13: Connecting to Remote Systems

Several utilities are available to enable you to transfer files between systems, to log into other systems on which you have an account, and to execute commands remotely. These features are described in this chapter.

Note that to use *any* of the communication methods with a VMS node, the VMS node must be running software to allow tcp/ip communication. All Fermilab VMS machines are licensed to run TGV's **MultiNet** software, which provides the necessary tcp/ip transport capability.

13.1 Transferring Files

13.1.1 ftp

ftp (File Transfer Protocol) can be used to transfer files to and from a remote system on which you have an account, and to manipulate the remote file system. The command format is:

```
% ftp [-n] [-i] [host]
```

If you specify ***host*** on the command line, you will be prompted for your (remote) username and password, and login will proceed. The option `-n` inhibits autologging on those systems on which it is enabled. The option `-i` eliminates prompting by the system, which can be tiresome when you're transferring many files.

Some commands available in **ftp** are described here; for a complete list, see the man pages for **ftp**.

`user` ***username password***	Identification for the remote system. If auto-log is enabled, you will be prompted automatically for the username and password. If you don't specify the password, it will prompt you.
`bin` or `binary`	Set transfer mode to binary image transfer.
`ascii`	Set transfer mode to ASCII (for text files).
`put` ***local_file*** [***remote_file***]	Transfer a local file to the remote system. If the remote filename is not specified, the local filename is used.
`mput` ***local_files*** [***remote_files***]	Similar to `put`, but you can use standard wildcard characters to transfer a series of files at a time.
`get` ***remote_file*** [***local_file***]	Transfer a remote file to the local system. If the local file name is not specified, it is given the name on the remote machine.
`mget` ***remote_files*** [***local_files***]	Similar to `get`, but you can use standard wildcard characters to transfer a series of files at a time.

help [*command*]	Display information about the meaning of ***command***. If no command is specified, **ftp** displays a list of the commands.
quit	Terminate the **ftp** session with the remote server and exit **ftp**.
open ***host***	Establish connection with remote host. Needed if ***host*** was not specified on command line.
close	Close connection with remote host and return to **ftp** prompt. Doesn't exit **ftp**.
dir [***remote_directory***] [***local_file***]	Print listing of the directory contents of the remote directory and optionally put the output in ***local_file***.
cd ***remote_directory***	Change working directory on the remote machine to ***remote_directory***.
lcd [***directory***]	Change working directory on the local machine. If no directory is specified, your home directory is used.
delete ***remote_file***	Delete the file ***remote_file*** on the remote system.
! ***shell_command***	Run a shell command without exiting **ftp**.

The following is an example **ftp** session where first one ASCII file, then a few binary files, are transferred from one UNIX machine to another. The username is unnecessary if the assumed one is correct. The password, as usual, is not displayed.

```
<fsui02> ftp cdfsga
Connected to cdfsga.fnal.gov.
220 cdfsga.fnal.gov FTP server ready.
Name (cdfsga:aheavey):
331 Password required for aheavey.
Password:
230 User aheavey logged in.
ftp> ascii
200 Type set to A.
ftp> put README
200 PORT command successful.
150 Opening ASCII mode data connection for 'README'.
226 Transfer complete.
local: README remote: README
121 bytes sent in 0.00091 seconds (1.3e+02 Kbytes/s)
ftp> bin
200 Type set to I.
ftp> mput appx*
mput appx_MH.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_MH.ps'.
226 Transfer complete.
local: appx_MH.ps remote: appx_MH.ps
63902 bytes sent in 0.33 seconds (1.9e+02 Kbytes/s)
mput appx_awk.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_awk.ps'.
226 Transfer complete.
local: appx_awk.ps remote: appx_awk.ps
56271 bytes sent in 0.24 seconds (2.2e+02 Kbytes/s)
mput appx_convrt.ps? n
mput appx_exmpl.ps? n
mput appx_impatient.ps? n
mput appx_login.ps? n
mput appx_ups.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_ups.ps'.
226 Transfer complete.
local: appx_ups.ps remote: appx_ups.ps
107683 bytes sent in 1.3 seconds (83 Kbytes/s)
ftp> quit
221 Goodbye.
<fsui02>
```

Note that the CERN utility **zftp** (**z** for Zebra) is available for transferring FZ and RZ files (ntuple files) between systems[1]. To use **zftp**, it must be installed on the server system, and you need to set up CERN library during the session (`setup cern`).

13.1.2 rcp

Another way to transfer files is with the **rcp** utility (stands for **r**emote **cp**y). You can use this command if both local and remote hosts support **rcp** protocol. The basic format is:

**`% rcp [`*options*`] `*file1 file2*

where *file1* and *file2* are the source and destination filenames, respectively; or

**`% rcp [`*options*`] `*file1 file2 ... directory*

where the files listed are to be copied to the directory. On a UNIX system the format of a remote directory or filename is ***hostname*:*path***, and the **MultiNet** implementation is consistent with VMS usage, namely ***hostname*::*path*** (note the double colon). If only filenames are given, they are interpreted relative to your home directory. Both the source and the destination may be on nodes other than the current machine.

The available options are:

`-p`	Give each copy the same modification times, access times, and modes as the original file.
`-r`	Copy each subtree rooted at *file1*, *file2*, etc. (for those that are directories); in this case the destination (last argument) *must* be a directory.

rcp does not prompt for passwords. Your current local user name must exist on the remote host and allow remote command execution by **rsh**. **rsh** is a utility that connects to a remote host and either executes a specified command or logs you on via **rlogin** (see section 13.3).

More information is available in the **rsh** and **rcp** man pages.

When you are using **rcp** between VMS and UNIX, you must protect special characters from local interpretation by enclosing them in quotes. An example command to copy a file `$HOME/prog1.c` from UNIX to the VMS node FNALV is:

```
% rcp prog1.c fnalv:"[.c]prog1.c"
```

Here is an example showing how to copy a file from remote UNIX node FNSG01 to the home directory on the VMS node (`$` represents the VMS prompt):

```
$ rcp fnsg01::"/usr/products/bufio/v1_00/src" bufio.src
```

If you are using the C shell family, **rcp** will not work if your `.cshrc` or `.login` on UNIX, or your `LOGIN.COM` file on a remote VMS host, executes interactive or output-generating commands. You should check for non-interactive access, and exit before such commands are executed, as is done for a UNIX system in the FUE default `.cshrc` file. The lines in `.cshrc` that check this are the following:

```
# Place any items that you want executed even for non-interactive use here
# Skip if not interactive shell
if ( $?USER == 0 || $?prompt == 0 ) exit
```

AFS

1. **zftp** is known to be likely to break in the AFS environment.

On a VMS system, check for **OTHER** (TCP/IP is not **NETWORK**):

```
$ IF (F$MODE () .EQS. "OTHER") THEN $ EXIT
```

Although you can specify a username and password with **rcp** (see the local man pages or **HELP**), you can also have a kind of "proxy login" by creating a special file (named `.rhosts`) on the remote machine containing the names of the systems and users on those system that should be allowed access.

13.1.3 The .rhosts File

To use many of the file transfer utilities, you need to set up what is known as a `.rhosts` file on the remote machine. This file must include a list of the names of the systems to be allowed remote access without login, and the usernames, where each system-username combination is separated by a space. If the username is left off, it is assumed to be the same as on the current system. If there are problems with name servers, you can be most assured that your address will be understood if you specify three lines for each connection requested:

1) one line with the complete Internet address,

2) one with the Internet number[1], and

3) one with the local name.

An example `.rhosts` file showing a single connection in the three specified formats:

```
fnsg01.fnal.gov
131.225.8.178
fnsg01 judy
```

When writing a `.rhosts` file on a UNIX host to allow access via remote commands (known as *r-commands*[2]) from a VMS cluster, you must include an entry for *each* cluster member and username from which you might access the UNIX host.

The following command line is an example of the command to copy a `FINGER.PLN` file from an account on FNALA to the file `.plan` in the same person's account on the UNIX node FNSG01. In order for this example to work, there must be a file named `.rhosts` on FNSG01 containing a line `fnala.fnal.gov`, and the username must be the same in both places.

```
$ RCP FINGER.PLN FNSG01::.PLAN
```

where **$** is the VMS prompt.

---

1. Use **nslookup** ***node*** to get the number. On VMS, you must enter **MULTINET NSLOOKUP** ***hostname***.
2. r-commands execute on remote hosts, and use `.rhosts` for authentication. They are variants of UNIX commands without the "r", and include for example **rsh** (**r**emote **sh**) and **rlogin** (**r**emote **login**).

13.2 Logging in to Other Systems

13.2.1 telnet

telnet is a remote terminal protocol which allows you to establish a connection to a login server at a remote site. **telnet** in the UNIX world is analogous to **SET HOST** in the VMS world. You can connect to other UNIX machines, or you can connect to VMS machines if the appropriate software is installed on the system (e.g., with **MultiNet** for which Fermilab has a site license). The format is:

```
% telnet [host]
```

If you include the host, you are prompted for the username[1] and password. If you do not include the host, you go into command mode. You can enter **help** at this point for help on the commands.

13.2.2 rlogin

Another way of logging into a remote computer from another system on the network is by use of the **rlogin** utility. One difference between **telnet** and **rlogin** is that if the communications package is configured for it and you create an appropriate `.rhosts` file (see section 13.1.3), you can log in to the remote system using **rlogin** without typing the password. The format from a UNIX machine is:

```
% rlogin rhost [-l username]
```

where:

rhost	is the remote host to which you want to log in
username	is the username if the name is different on the remote system than the originating system

From a VMS machine, the format is:

```
$ RLOGIN /USERNAME=username rhost
```

If your system uses AFS and you experience permission problems after using **rlogin**, run **pagsh** followed by **klog**. Please see the discussion of these commands in section 7.3.1.

13.3 Executing Commands Remotely: rsh

You can execute commands on a remote system and have the output displayed on your terminal with **rsh**. **rsh** stands for **r**emote **sh**ell. In order for this to work, there must be an appropriate `.rhosts` file on the remote machine (see section 13.1.3).

The format is:

```
% rsh rhost [-l username] command
```

---

1. Some systems may not prompt you for a username; they will use your current login id. If you need to log in with a different username, enter a carriage return at the password prompt. Then the system will prompt you for a username.

where:

rhost	is the remote host on which the command is to be executed
username	is the username on the remote system if different than the originating system
command	is the command to be executed on the remote system

The command may be a single command or a shell script to be executed on the remote system. Metacharacters in the command must be quoted for proper interpretation on the remote machine.

The **MultiNet** implementation on VMS is:

`$ RSHELL[/INPUT=`*infile*`][/OUTPUT=`*outfile*`][/USERNAME=`*username*`]-` ***hostname commandline***

The **MultiNet** implementation does not handle metacharacters like the UNIX implementation does.

The **`/INPUT`** and **`/OUTPUT`** options in the **MultiNet** implementation allow you to execute a task on a UNIX machine which reads and writes files on VMS.

If the command is omitted, you will be logged in to the remote host using **rlogin**.

Chapter 14: Batch Processing Environment

In this chapter we provide introductory information on **LSF** (Load Sharing Facility), the standard batch processing system at Fermilab, and on **fbatch**, the locally-written interface to **LSF**. We also list the related software components that can be used with **LSF/fbatch**.

You should be able to run and manipulate most batch jobs easily after reading this chapter.

14.1 The Standard Batch System at Fermilab: LSF

LSF, developed by Platform Computing, is a general purpose resource management system that unites a group of UNIX computers into a single system to make better use of the resources on a network. The single system is referred to as a cluster. **LSF** collects resource information from all nodes in the cluster, and uses it to allocate the available host machines for execution of batch jobs.

LSF distinguishes between client machines and server machines. A job can be submitted from either type, but run only on a server (a host). Under **LSF**, jobs that are run remotely behave just like jobs run on the local host. Even jobs with complicated terminal controls behave transparently to the user as if they were being run locally.

LSF is fully documented on-line; see the *LSF User's Guide*. To find it, select *Batch Services* from the Computing Division home, and look under *LSF v3.0 Documentation*. There is also a link to it in the Computing Division document database under **fbatch**.

For the purposes of this chapter, a batch job (also called simply a *job*) is any UNIX executable that is submitted to the **LSF** batch system. Job control information (e.g., name of executable, queue, required resources, and so on) is passed to **LSF** via command line arguments supplied when submitting a job.

14.1.1 Job Queues

Batch jobs are submitted to **LSF** via *job queues*. **LSF** administrators generally configure job queues to control host resource access according to user and application type. A queue can be defined to use a particular subset of the hosts in the **LSF** cluster; the default is to use all hosts.

Each queue represents a different job scheduling and control policy. Users select the job queue that best fits each job. All jobs submitted to the same queue share the same scheduling and control policy. There is a **nice** value associated with each queue (see section 5.5.1), and jobs submitted to a queue are automatically "re**nice**d" accordingly.

14.1.2 Load Monitoring on Hosts

LSF monitors the load of each host in the batch cluster by comparing the values of several built-in load indices against the allowable load thresholds defined by the **LSF** administrator. A load index is simply a measurement of the processing load on a batch host. On an overloaded host, batch jobs can begin interfering with each other or with interactive jobs. Therefore, **LSF** begins suspending jobs on a host when it becomes overloaded (i.e. when one or more load indices exceed the predefined *suspension threshold*). **LSF** resumes any suspended jobs once all the load indices read below the *release threshold.*

If a job queue has been defined with a time window (measured in real time), **LSF** suspends any jobs running on that queue when the current time falls outside of the window. These jobs get released when the time window reopens.

14.1.3 Host Selection

The resources available for processing **LSF** jobs on each host are defined by an **LSF** administrator. Only nodes having resources that match or exceed the resource requirements of a given job are potential hosts for that job. **LSF** compares the resource requirements specified for the job against the load on each of these nodes, and chooses the most favorable host.

If no resource requirements are specified for a job, a host of the same model and type as the machine on which the job was submitted is chosen.

14.1.4 Job Priority

LSF schedules, suspends, and releases submitted jobs by balancing job priority and available resources. Job priority is governed by several factors:

- the options and arguments specified on the command line during batch submission
- the priority of the queue on which the job was submitted; according to **LSF**'s FCFS (first come first serve) protocol
- the number of shares that a job has used, according to the FSS (Fair Share Scheduling) protocol. A *share* is a portion of the resources available on the host or hosts; queues may be defined to limit the number of shares jobs can utilize.

When a host's suspension threshold is reached, **LSF** suspends lower priority jobs first unless the scheduling policy associated with a particular job dictates otherwise. A suspended job can later be resumed by **LSF** if the host's release threshold is again reached (or, if the suspension was due to a time window, as mentioned above, the job resumes when the time window reopens).

LSF does not override the UNIX scheduler.

14.2 Local Interface to LSF: fbatch

The **UPS** product **fbatch** supplies the commands that you enter to run and manipulate batch jobs. It is a set of locally-written shell scripts and C programs that provides a wrapper around **LSF**, and thus characterizes the batch processing environment on a cluster. **fbatch** is installed on many Fermilab systems, including FNALU. **fbatch** supports all the **LSF** batch functionality, and provides in addition:

- a consistent interface to **LSF** commands, regardless of UNIX flavor
- supplementary commands not directly supported by **LSF**
- an interface to the Fermilab products **spacall** and **needfile** (see section 14.3)

- AFS token renewal at job execution time
- a remote shell facility allowing commands to be executed from any host; if the host is neither a client nor a server, **fbatch** issues the **LSF** command via the **rsh** facility (see section 13.3)
- case-insensitive resource names

You need to run the command **`setup fbatch`** before accessing **fbatch** commands.

When you setup **fbatch**, you will also be able to access the man pages for these commands. Running **`man fbatch`** returns a list of all the commands supported under **fbatch**. For a complete description of the **fbatch** product, refer to the *fbatch User's Guide* (PU0152).

Several of the **fbatch** commands are illustrated below, organized by function. For complete information on each of the commands, see the man pages.

14.2.1 View Host Information

To see which hosts and resources are defined in your cluster, you can issue the command:

```
% fbatch_hostinfo
```

The configuration information returned includes: host name, host type, host model, CPU factor, number of CPUs, total memory, total swap space, whether the host runs **LSF** servers or not, available resources denoted by resource names. The host name, host type, and host model fields are truncated if too long. The CPU factor is used to scale the CPU load value so that differences in CPU speeds are considered by LIM[1]. The faster the CPU, the larger the CPU factor.

The output is returned in this format:

```
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
fsgi02          SGI R4400Ch2  84.0    16   511M  2755M    Yes (irix any fsgi02)
fsui02       SUNSOL ULTRA167  93.0     4   320M   889M    Yes (sparc any sun fsui02)
fibb01          AIX     I560  39.0     1   192M   400M    Yes (aix any fibb01)
fncl10          AIX     I370  49.0     1   128M  1136M    Yes (aix any clubs fncl10)
fibi01          AIX     I590  62.0     -      -      -     No ()
fsgi01          SGI   I4D420  30.0     -      -      -     No ()
```

14.2.2 View Queue Information

The **`fbatch_queues`** command lists the available **LSF** batch queues:

```
% fbatch_queues
```

The output returned is in the following format. A dash (`-`) in any entry means that the column does not apply to the row. In this example some queues have no per-queue, per-user or per-processor job limits configured, so the `MAX`, `JL/U` and `JL/P` entries are dashes. The man page describes each of the fields.

1. Load Information Manager (LIM) is a daemon process that keeps track of the load indices.

```
QUEUE_NAME      PRIO      STATUS      MAX  JL/U JL/P JL/H NJOBS  PEND  RUN  SUSP
test_queue        99   Open:Active     -    -    -    -     0     0    0     0
e831_long         16   Open:Active     1    1    -    -     0     0    0     0
e831_short        14   Open:Active     -   10    -    -     0     0    0     0
30min             10   Open:Active     -    5    -    -     1     1    0     0
30min_disk        10   Open:Active     -    5    -    -     3     3    0     0
4hr                8   Open:Active     -    5    -    -     2     0    1     1
4hr_disk           8   Open:Active     -    5    -    -     5     2    3     0
12hr               6   Open:Active     -    5    3    -     3     0    3     0
12hr_disk          6   Open:Active     -    5    2    -     7     4    3     0
1day               4   Open:Active     -    5    1    -     0     0    0     0
1day_disk          4   Open:Active     -    5    1    -     7     0    7     0
4day               2   Open:Active     -    5    0    -    33    17   12     4
```

You can submit jobs to a queue as long as its `STATUS` is `Open`. However, jobs are not dispatched unless the queue is `Active`.

14.2.3 Submit a Batch Job

The **`fbatch_sub`** command is used to submit a job to the batch system. The most common arguments used are **`-q`** (queue name), **`-R`** (resource requirements), **`-o`** (stdout redirection), **`-e`** (stderr redirection), and **`-N`** (notify via email).

As an example, here we submit a script called `myjob` to the `4hr` queue, specify an IRIX host, and request notification. The stdout is redirected to `myjob.out`, and the stderr to `myjob.err`:

```
% fbatch_sub -N -q 4hr -o myjob.out -e myjob.err -R "irix" myjob
```

AFS

On systems running AFS, **fbatch** will prompt you for your AFS password[1]:

```
Enter AFS Password...
Reenter AFS Password...
fbatch_sub executing LSF command locally on fsui02....
```

When your job begins, you will automatically receive a renewed AFS token on the execution host.

14.2.4 Monitor Submitted Batch Jobs

fbatch provides several commands that allow you to monitor your job. The usage examples below use a sample job number 1022:

Display a listing of running jobs

```
% fbatch_jobs
```

If no options are supplied, the list will contain only your running jobs. To see *all* running jobs, use the **`-u all`** option. Output is returned in this format:

```
JOBID USER     STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
 1022 aheavey  PEND  30min      fsui02                  sleep1      Sep 10 09:56
```

Display the stdout and stderr of a job

```
% fbatch_peek 1022
```

The format of the output varies according to the files.

1. Your password gets encrypted by **fbatch** using PGP.

Display history information about a job

```
% fbatch_hist 1022
```

Output is returned in the format:

```
Summary of time in seconds spent in various states:
JOBID USER    JOB_NAME   PEND    PSUSP   RUN     USUSP   SSUSP   UNKWN   TOTAL
 1022 aheavey sleep1     7       0       35      0       0       0       42
```

14.2.5 Control Submitted Batch Jobs

For jobs that are in a queue awaiting execution, **fbatch** provides commands to move jobs within the queue, and to modify the resource requirements of the job. The usage examples below use a sample job number 1022:

Move Job within Queue

Move job to the bottom of the queue:

```
% fbatch_bot 1022
```

Move job to the 2nd position from the top of the queue:

```
% fbatch_top 1022 2
```

The **fbatch_bot** and **fbatch_top** commands, above, move jobs within queues *relative to the user's own jobs*. You cannot move your job ahead of another user's job with these commands.

Change Job Parameters

Change the resource requirements of job:

```
% fbatch_modify -R aix 1022
```

Migrate a batch job to another host:

```
% fbatch_mig -m newhost 1022
```

Suspend, Resume, or Kill a Job

Suspend (stop, but do not cancel) job:

```
% fbatch_stop 1022
```

Resume job:

```
% fbatch_resume 1022
```

Cancel job:

```
% fbatch_kill 1022
```

14.3 Related Software Components

This section describes other software components that can be used with the **LSF/fbatch** batch system.

needfile

needfile is an interface to the **HPSS/Unitree** central mass storage system. The **needfile** system provides two functions. If the user's data does not yet exist in **HPSS**, **needfile** tells the system to copy the data there from its original medium. **needfile** then provides access to this data for the user. This data remains in the HSM based on a least-recently-used policy. There is no guarantee how long data will remain in the HSM.

To access **needfile**, first run the command `setup nt` (setup not required if accessed via **fbatch**).

A **needfile** reference manual exists on the Web. To find it, start on the Computing Division home page, select *Mass Storage* under **Systems and Networking** and then *fmss* to find **needfile tool**. There is also documentation there on HPSS.

spacall

The **spacall** utility (space allocator) provides scratch disk storage for a job. **spacall** is invoked under **fbatch** by submitting a job to a specially defined queue; for example, on FNALU the `*_disk` queues have been configured for it. The path to the scratch space is stored in the environment variable *$CLUBS_WORKDIR*.

Chapter 15: Tape Handling

In this chapter we discuss the principal tape handling software and facilities available at Fermilab. Start-up information for running and monitoring **OCS** tape mounts is provided, and the **OCS** X interface is introduced. Several tape I/O packages are briefly described.

OCS manages tape drive allocations and operator-assisted mounts, whereas **RBIO** and **DAFT** are tape I/O packages. **FTT** is a low-level C subroutine library common to **OCS**, **RBIO** and **DAFT**; it can also be used directly for tape I/O and tape drive testing. **FMB** includes features for both tape mounting and I/O. These are all **UPS** products, and are accessible in the standard way via the `setup` command.

Since many groups and experiments have customized their tape-handling routines, some or all of the applications described here may not be available or appropriate for you. Contact your group leader to find out what procedures have been established for your group.

15.1 Operator Communications Software (OCS)

OCS is a package that performs and manages tape drive allocations, operator-assisted tape mounts and tape drive use statistics. Its logical-to-physical tape device name translation helps not only human communication, but hides many platform-specific idiosyncrasies from users. The latest version as of this writing is v3.0.

OCS is not a tape I/O package. It works well *with* such packages (e.g., **RBIO**, **FMB**, **DAFT**).

OCS functions are available via three separate interfaces:

- FORTRAN/C library of subroutines
- Command line tools
- X Motif tools

The features available in each of these interfaces overlap to a great extent. However, since the different interfaces are by their nature geared towards different uses, the functionality is not 100% duplicated across them. The FORTRAN/C subroutines provide the most flexibility and functionality for users requiring multiple mount requests, the command line tools are generally used in shell scripts, and the X interfaces are very useful for monitoring tape drive statistics.

The X interfaces are fairly intuitive, so we give you enough information to bring them up (section 15.1.2), but we do not describe them in detail.

The **OCS** functions are fully described in the *OCS Reference Guide*, document number GA0012, available on the Web. The reference guide and other documents are also available wherever **OCS** is installed and setup, in the directory `$OCS_DIR/doc`.

15.1.1 OCS Basics

Monitoring Tape Mounts

During the course of your work, you may need to monitor different aspects of the job and possibly contact the operators. **OCS** provides functionality to perform these job management activities. For your convenience, we list the appropriate command next to the function in the table below, and indicate whether you can perform the function using one of the X interfaces. You will need to see the *OCS Reference Guide* for usage information on these commands and for the associated FORTRAN/C subroutines.

Function	Command	X interface
Mark a tape drive broken so that it will not be allocated until it is repaired	**ocs_broken**	**xocs**
Log statistics as to how much the drive was used	**ocs_report_stat** **ocs_init_stat**	
Send an attention message to the operator	**ocs_message**	
Send a request to the operator to run a cleaning tape through a tape drive	**ocs_clean_it**	
Display status of tape drives in the **OCS** database	**ocs_tape**	**xocs**
Display pending mounts	**ocs_pending**	**xocs** **xtapeview**
Display tape drive statistics	**ocs_devstat** **ocs_stats**	**xocs**
Display tape mount log	**ocs_mrlog**	**xocs**
Display tape drives that may need to be cleaned	**ocs_clean_list**	**xocs**

Sample Tape Mounting Process

Here is a sample tape mounting process using the command line tools. Read through it to see what steps are involved and what kind of response to expect from the system at each step. Note that the **OCS** commands come with many options that are not shown here.

First, display the list of available tape drives:

```
% ocs_tape

   HOSTNAME DEVICE   TYPE     ALLOCATED STATUS  VSN            USERNAME UID   AUTH
   bastet   dumdlt4  DLT4000  allocated working -              root     0       n
   bastet   isis2    EXB-8505 unalloctd working -              -        -       n
   bastet   horus    EXB-8200 unalloctd working -              -        -       y
```

Request allocation of a tape drive so no other user may access it (notice only one device shows authorization as "yes"):

```
% ocs_allocate

   bastet horus
```

Send a request to the operations staff to mount a tape on the drive you have allocated:

```
% ocs_request -t horus -w -v FGMS04

    ocs_request: success
```

Verify that the tape was mounted correctly:

```
% ocs_check_label -t horus -w -v FGMS04

    ocs_check_label: Success
```

Set the tape drive characteristics according to your needs (we recommend that you always do this; don't assume the drive has been left in any particular state):

```
% ocs_setdev -t horus
```

Request the appropriate device file for reading and/or writing the tape according to the characteristics you've set:

```
% ocs_devfile -t horus

    /dev/rmt/tps0d3nrv
```

Perform your task on the loaded tape. Normally this involves running a program that calls tape I/O routines from **RBIO** or another I/O package. For simplicity in this example, we just use the UNIX **dd** utility to get the tape contents (we have loaded an ANSI initialized tape with no data):

```
% dd if=/dev/rmt/tps0d3nrv conv=unblock cbs=80

    VOL1FGMS04              ftt                                     4
    HDR1
    0+2 records in
    0+1 records out
```

A useful feature to include in this example is the device statistics function. It provides more detailed information for the end user than the X interface implementations, which were designed more for administrative purposes.

```
% ocs_devstat -t horus

    ------------------------------------------------------------------------------
    Collecting Tape Drive Statistic ------------------- Thu Oct  2 12:43:27 1997
    ------------------------------------------------------------------------------
    Host Name             = bastet
    Device Name           = horus
    Device Filename       = /dev/rmt/tps0d3
    Device Type           = EXB-8200
    Controller            = SCSI
    Vendor Id             = EXABYTE
    Product Id            = EXB-8200
    Firmware Id           = 268N
    Serial Number         = -
    Number of Hours On    =          -1   In Motion =          -1
    Count/KBytes    Read  =        2200       Write =          -1
    Errors          Read  =           0       Write =          -1
    Comp Ratio      Read  =          -1       Write =          -1
    Compresion            = NO
    Density               = 8200
    Media Type            = 133
    Block Size            = 0
    Block Total           = 2294048
    Count Origin          = Exabyte_Extended_Sense
    Remain Tape/KBytes    = 2290569
    SCSI Sense Code       = 0
    SCSI ASCQ             = 0
    Track Retry           = -1
```

```
Stop/Start Count     = -1
SCSI Test Unit Ready = 0
SCSI Sense Key       = NO_SENSE
-------------------------------------------------------------------------------
Tape Not Present:  N | Write Prot:   N | Clean Bit:     N | Drive Cleaned: N
Beginning of Tape: N | At File Mark: N | End of Media:  N | End of Tape:   N
Ready Bit:         Y | Power Fail:   N | SCSI ILI Bit: N |
-------------------------------------------------------------------------------
```

Dismount the tape (i.e. rewind and unload it):

```
% ocs_dismount -t horus
```

```
ocs_dismount: Success
```

Finally, deallocate the tape drive so that someone else can use it:

```
% ocs_deallocate -t horus
```

```
ocs_deallocate: Deallocated :horus
```

Tape Mounts with Batch Jobs

Most of the time, users run tape mounts in conjunction with batch jobs. We strongly recommend that you include the tape mount allocation, mount request, dismount and deallocation *within* your batch job. If you don't, you risk preventing others from using the tape drive while your job is waiting to run and after it has finished. A **`-q`** option is provided for the **`ocs_allocate`** command to allow you to queue for tape drive allocation so that your job won't fail if a drive isn't immediately available when it starts to run.

Here is a sample batch job script that incorporates these recommendations:

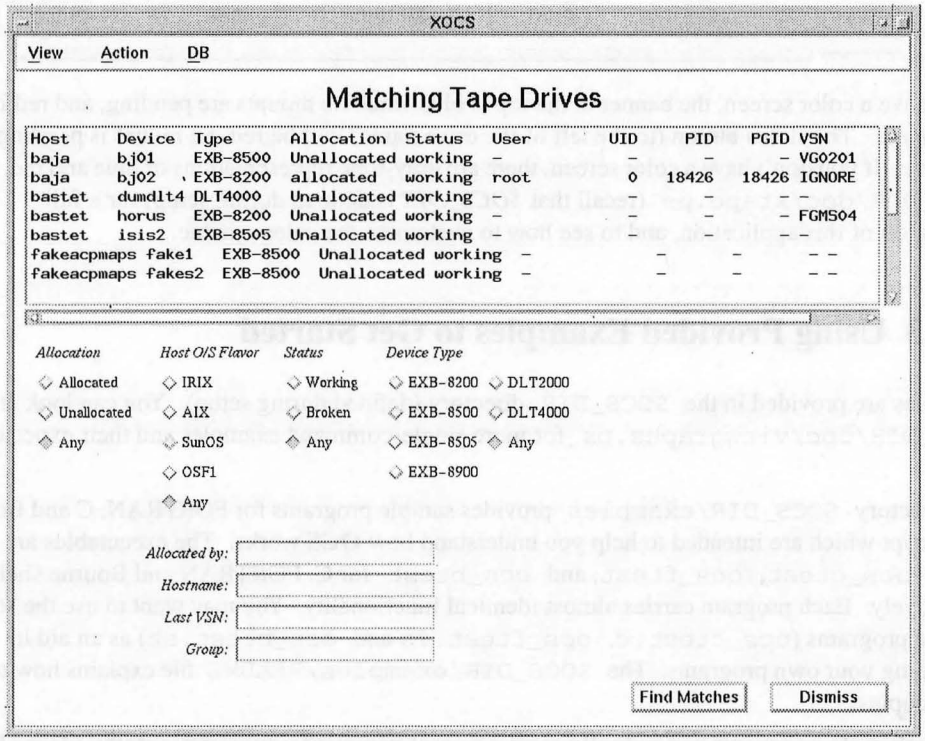
```
#!/bin/csh
setup ocs
set td=`ocs_allocate -q -h localhost -d exabyte_850x | cut -f2 -d" " -`
if ( $status != 0 ) then
     echo ocs_allocate failed with message: $td
     exit 1
endif
set drive=`echo $td | cut -f2 -d" " -`
ocs_request -t $drive -v fr3147 -r
if ( $status != 0 ) then
     echo ocs_request failed
     ocs_deallocate -t $drive
     exit 1
endif
set dfile=`ocs_devfile -t $drive`
if ( $status != 0 ) then
     echo ocs_devfile failed with message: $dfile
     ocs_dismount -t $drive
     ocs_deallocate -t $drive
     exit 1
endif
ocs_setdev -t $drive -v -d 8500
if ( $status != 0 ) then
     echo ocs_setdev failed
     ocs_dismount -t $drive
     ocs_deallocate -t $drive
     exit 1
endif
setenv MY_DEVFILE_VAR $dfile
e831job.run
ocs_dismount -t $drive
ocs_deallocate -t $drive
exit 0
```

15.1.2 The OCS X Interfaces

Remember that your *DISPLAY* environment variable needs to be set properly for X windows applications (see section 9.2).

xocs

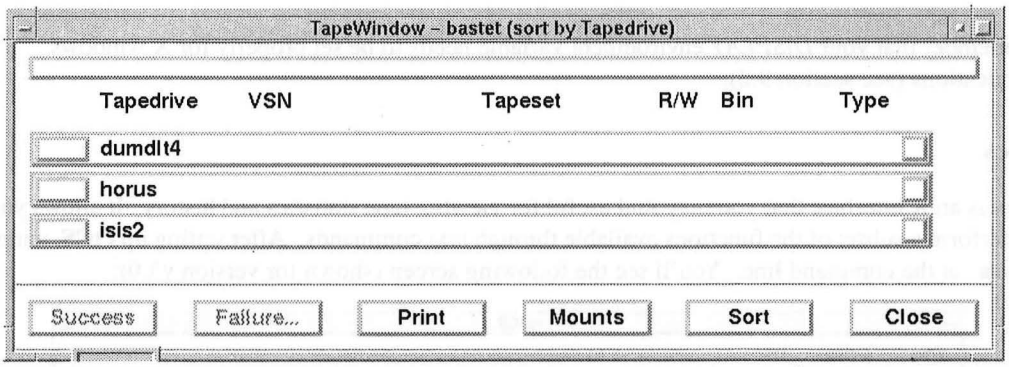
xocs is an X interface that you may find useful for viewing tape statistics and history. It allows you to perform a subset of the functions available through text commands. After setting up **OCS**, enter **`xocs`** at the command line. You'll see the following screen (shown for version v3.0):



The menu options under View and Action include most of the features you will need.

xtapeview

Another X interface to **OCS** which allows you to view pending mount requests is **xtapeview**. As a user, you are not permitted to respond to mount requests, only view them. Invoke this interface with the command **`xtapeview`**. A window will appear from which you can choose a node. Click on the node you want, then you'll see a window like the one below which shows the drives associated with that node, and any pending mounts.



If you have a color screen, the banner at the top is blue when no mounts are pending, and red if one or more are. The status button (to the left of the drive name) will be red if a mount is pending on that drive. If you don't have a color screen, there are gray-tone representations of blue and red. See `$OCS_DIR/doc/xtape.ps` (recall that *$OCS_DIR* is defined during setup) for a full description of this application, and to see how to customize the color scheme.

15.1.3 Using Provided Examples to Get Started

Examples are provided in the `$OCS_DIR` directory (defined during setup). You can look at `$OCS_DIR/doc/viewgraphs.ps` for more single-command examples and their expected output.

The directory `$OCS_DIR/examples` provides sample programs for FORTRAN, C and Bourne shell script which are intended to help you understand how **OCS** works. The executables are named `ocs_ctest`, `ocs_ftest`, and `ocs_btest` for C, FORTRAN and Bourne shell, respectively. Each program carries almost identical functionality. You may want to use the source of these programs (`ocs_ctest.c`, `ocs_ftest.fs` and `ocs_btest.sh`) as an aid in developing your own programs. The `$OCS_DIR/examples/README` file explains how to use the examples.

These example programs can actually send a mount request to operations staff: please keep this in mind if you experiment with OCS on Fermilab Computing Division systems such as FNALU.

15.2 Raw Buffered I/O (RBIO)

The **RBIO** (Raw Buffered I/O) library provides an I/O interface that is FORTRAN- or C-callable, medium-independent, and UNIX flavor-independent. It allows your FORTRAN or C program to perform buffered I/O on a file object, which may be a regular file or a labeled tape. The **RBIO** library is used in the compilation, linkage, and execution phases. The actual reading/writing level is at the primitive logical record level; **RBIO** does not understand record contents.

RBIO is documented in the *RBIO User's Guide*, document number SU0033.

15.3 DAta From Tape (DAFT)

DAFT (DAta From Tape) consists of a set of routines written by the DART collaboration to provide event-level reading and writing from DART-formatted records to/from disk and tape. Information on DART can be found under *Projects & Working Groups* on the Computing Division home page.

DAFT performs the following functions:

- retrieve data written on tape or disk in the format used by DART
- write data on tape or disk in the DART format

The routines in **DAFT** are widely portable, and currently run on the supported UNIX platforms IRIX, Solaris, AIX and OSF1. They offer a FORTRAN interface.

DAFT was written in two layers to allow for a lower level that is easily portable. Therefore, two sets of access routines are provided:

- a "low-level" set that embeds the different levels of platform-dependence and delivers records to the higher level routines
- "high-level" routines that unpack events from the records and provide the event interface

The DART on-line format packs events into logical records, where each record may contain more than one event, in order to keep the drives streaming at full speed. The routines handle large events spanning records (the maximum record length is 64 Kbytes), and correct the byte ordering of data on platforms of different *endianship*.

For more information on **DAFT**, see the *DAFT (DAta From Tape) User's Guide*, document number PN504.

15.4 Fermi Tape Tools (FTT)

The Fermi Tape Tools (**FTT**) product is a low-level C subroutine library. It provides:

- basic tape I/O
- support for obtaining and tracking available tape statistics
- a set of informational and support subroutines for tools that allocate and control tape usage (i.e. **OCS**)

FTT has been designed to provide platform- and drive-independent I/O on the UNIX platforms supported at Fermilab. It has also been designed to be portable to other platforms and drives. In addition to its use as a transparent layer of software in several products (namely **DAFT**, **OCS**, and **RBIO**), the **FTT** subroutine library can be called directly by user-written programs. **FTT** also includes an interactive test program which, among other things, can verify that tape drives are behaving properly.

FTT is documented in the *Fermi Tape Tools Library User's Guide*, document number PU0236.

15.5 Fermi Modular Backup (FMB)

The Fermi Modular Backup (**FMB**) system is a locally-written set of utility scripts used for backup, restore, and other similar administrative tasks. It was designed for use on the supported UNIX flavors. **FMB** uses short, simple-to-write *modules* to provide different tape rotation schemes, tape mount systems, archive formats, and so on. It also supports backup of remote machines, and/or remote tape drives.

See the *Fermi Modular Backup System User's Guide*, document number PU0164.

Chapter 16: Software Development

This chapter gives an introduction to UNIX software development tools in common use at Fermilab, providing information on:

- Supported languages
- Compiling and linking in C, C++ and FORTRAN
- Debugging

We do not include a discussion of general programming here, but rather, aspects of software development particular to UNIX. You will need to reference the man pages and the vendors' manuals for more system-specific details.

Useful documents relating to FORTRAN, C, and C++ programming can be found under **Software Development** on the *UNIX Resources* Web page.

Many software development subjects of interest to Fermilab users are beyond the scope of this manual. Among them are:

- Object Oriented programming techniques including NextStep
- CASE tools
- Neural Network methods
- Lattice Gauge techniques used in ACP-MAPS, CANOPY, et.al.
- Data Mining methods available in the CAP facility.
- Many excellent commercial tools for building Graphics User Interfaces, debugging, migrating and analyzing performance of user code, building Database interfaces, etc.

16.1 Overview of Programming Languages and Tools

This is a short overview of some common programming languages. Our aim here is to give you a general idea of what tools are available, and how they can be used. The list is ordered from low to high level, and indicates common uses:

Assembler	not used at Fermilab
C	system services, user interface, general utilities
FORTRAN	FORmula TRANslation (physics calculations)
C++	object oriented general programming
Perl	interpreter, general purpose
Python	object-oriented and/or general purpose interpreted scripting language
Tk	interpreter, GUI interfaces

Assembler

Traditionally, assembler has been needed for a couple of reasons:

- Access to system services and hardware
- Tuning program efficiency

Assembler programming is not generally necessary or desirable on the RISC based UNIX systems presently in use. RISC system performance is so heavily dependent on pipelining and various caches that it is extremely difficult, if not impossible, for an individual to write more efficient assembler code than is generated by the higher level language compilers. The C language provides all the hardware access and system service capabilities traditionally provided by Assembler.

C

As noted above, C has filled the programming niche traditionally occupied by Assembly language. In addition to its normal use as a high-level programming language, C can act as a universally portable Assembler.

Because the C language was created, has evolved, and has become standardized hand-in-hand with the UNIX operating system, it is the language of choice for applications involving system services and user interfaces. Using C is discussed in several of the following sections.

Now that an ANSI C standard exists and is widely implemented, portability of C code is much improved. ANSI C compilers are the default on most Fermilab systems. Likewise, adoption and availability of POSIX standards for operating system services has greatly improved program portability.

An excellent reference book for C programming is *The C Programming Language* by Kernighan and Ritchie published by Prentice-Hall.

FORTRAN

FORTRAN remains the most effective language for mathematical calculations. This is due partly to decades of research which has produced highly efficient optimizing compilers, and partly to the millions of lines of tested, portable code already in use.

C++

C++ adds object oriented programming constructs to the C language. At the risk of oversimplifying, it seems that C++ is substantially harder to learn and to use for writing new programs, but the resulting programs are much better structured, more maintainable, and more shareable than traditional C or FORTRAN programs.

An additional advantage of C++ is the availability of class[1] libraries. A Standard Template Library will come with most compilers soon. This library will contain many useful low level classes for such things as strings and streams I/O. Also, some vendors include other commercial class libraries as added value to their compilers. In addition to the vendor-supplied versions, C++ is available as a part of Gnu C. This has increased its popularity.

The C++ language standard is (still!) in the process of adoption by the ANSI and ISO standards committees. Since the C++ standard has not been finalized (although it changed significantly in December 1996), there are at this time (November 1997) cross-platform porting issues. None of our major vendors (SGI, IBM, GNU) are yet providing a draft-standard-compliant compiling and runtime toolkit. The safest bet for generating portable C++ code is to avoid using newer features such as exceptions and templates, which may be implemented differently (or not at all) by the

1. A *class* is similar to a *structure definition* in C.

various C++ compiler vendors, at least until the vendors catch up with the standard. As an alternative, use the `g++` command to run the Gnu integrated C/C++ compiler on all platforms. Using the `g++` command instead of `gcc` gives you appropriate C++ linking.

For documentation, you may refer to:

- the `CC` (upper case), `gcc` and `g++` command man pages
- *The C++ Programming Language*, Addison-Wesley, by Bjarne Stroustrup

Perl

perl is installed at Fermilab as part of the **shells** product. The man page for **perl** gives a good brief description:

> **perl** is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

There is an excellent text published by O'Reilly & Associates, Inc. on **perl**.

Python

Python is an interpreted, interactive, object-oriented programming language often compared to **Tcl**, **Perl**, **Scheme** and **Java**. **Python** combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems (e.g., **X11**, **Motif**, **Tk**, **Mac**, **MFC**, **STDWIN**). New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that require a programmable interface.

An on-line document for **Python** is available in the CD documentation database.

Tk

Tk is an **X11** toolkit that provides the Motif look and feel, and is easy to use for building graphical interfaces largely because it is built on an interpreted language. It can be used with a variety of languages including **Tcl** (**Tk** used to be *solely* implemented using **Tcl**), **Perl**, and **Python**.

The best reference for **Tk** is the book *Tcl and the Tk Toolkit*[1], by John K. Ousterhout, published by Addison-Wesley. The `README` file under the `tk` directory in `$TK_DIR` (created during setup) points to a draft of this book. Also see the man pages for information on these languages.

Other Languages and Language-Related Tools

Other tools exist that are commonly used as languages in the appropriate context. These include, for example, the various UNIX shells (as discussed in section 4.4), and **awk**, **sed**, **yacc**, and **lex**, for which O'Reilly & Associates, Inc. provides excellent texts.

There are many other languages which are not widely supported or are not in general use at Fermilab. These include Pascal, Modula-2, Lisp, Forth, Bliss, and others. We do not discuss them in this document.

---

1. Some publishers' catalogues use an ampersand (&) rather than the word "and"; check both in database searches.

16.2 Introduction to C and FORTRAN on UNIX

16.2.1 The C Compiler: cc

cc (lower case) is the vendor-supplied C compiler command on all Fermilab-supported UNIX systems (except LINUX, where it is **gcc**). ANSI C compilers are the default on most Fermilab systems. To compile one or more C source files (***filename.c***), you run the **cc** utility. **cc** automatically invokes the link editor **ld** unless the option **-c**, which explicitly suppresses linking, is used.

16.2.2 The FORTRAN Compiler: f77

All the Fermilab supported UNIX systems have good FORTRAN 77 compilers, which provide some minimal extensions. These compilers also recognize most DEC-supported VAX-FORTRAN extensions.

f77 is the FORTRAN compiler command on all Fermilab-supported UNIX systems (except LINUX, where it is **g77**). The **f77** command controls both compilation and linking functions automatically using appropriate FORTRAN runtime libraries. **f77** can produce object modules, partially linked objects, or executable programs, as appropriate. The option **-c** explicitly suppresses linking.

Note that on AIX, to get help on **f77** you need to type **man xlf** rather than **man f77**.

16.2.3 C and FORTRAN Compiling Basics

UNIX compilers, including **f77** and **cc**, generally use both filename extensions and the file content to determine how to handle the files listed on the command line. The commonly used extensions are:

For C

c	C source, e.g., myfile.c

For FORTRAN

f	FORTRAN source, e.g., myfile.f

For both

o	object file, e.g., myfile.o
a	archive library, e.g., mylib.a
(none)	executable image, e.g., myfile

For historical reasons the UNIX linkers produce by default an executable named a.out; the option **-o *filename*** is available to override this default.

We list here some basic compiling and linking examples. In these examples, we use a source file named foo.c, where c is the standard extension for C. In all instances shown here c can be directly replaced by f, and **cc** by **f77**, for FORTRAN.

To produce:	... enter the command:
foo.o object module	**cc -c foo.c**
foo executable from source	**cc -o foo foo.c**

`foo` executable from source + object file

```
cc -o foo foo.c myobj.o
```

`foo` executable from source + library

```
cc -o foo foo.c $CERN_DIR/lib/libmathlib.a
```

`foo` executable from source + X11/Motif (standard system libraries)

```
cc -o foo foo.c -lXm -lXt -lX11
```

The options used with the **cc** and **f77** commands are discussed in later sections.

16.2.4 Linking Order

Most UNIX linkers process source, object and library files in the order that they occur on the command line. Backward library references, from a file to an earlier library, will not be satisfied, except under AIX. It may be necessary to list a library more than once for successful linking.

AIX loads the entire content of all libraries into memory, eliminating ordering problems, but potentially creating memory usage problems. Listing a library more than once is unnecessary, and positively undesirable under AIX.

16.2.5 Displaying Active Options

You may wish to know which options are active for a given compilation, in order to verify that the defaults are what you expect. Each platform seems to provide this information somewhat differently:

Platform	Option	Result
AIX	`-q listopt`	Source listing with options, to file `*.lst`
IRIX	`-v`	Options and other details, to `stdout`
OSF1	`-V`	Source listing with options, to file `*.l` (FORTRAN only)
SunOS	not available	

16.2.6 Option Passing

At each stage of compilation, any unrecognized command line options are passed on to the next stage. Options that could be valid for more than one stage can be explicitly passed to a particular stage. To direct an option to a specific phase of compilation or linking, use the option **-W** (for AIX, IRIX, OSF1) or **-Qoption** (for SunOS). The phase is identified by the letter immediately following the **-W**. Thus, for example, **-Wl,-m** tells **f77** or **cc** to pass the option string **-m** to the **l** (link) phase. The **f77** command line might read:

```
% f77 -Wl,-m foo.f
```

for AIX, IRIX, or OSF1, and

```
% f77 -Qoption ld -m foo.f
```

for SunOS (where **ld** is the loader program)

16.3 Introduction to C++ on UNIX

`CC` (upper case), `g++` and `gcc` are all C++ compiler commands on the Fermilab-supported UNIX systems that provide C++ (see section 16.1 for a brief discussion). Just as C++ is a superset of C, the C++ compilers are very similar to C compilers in that their options are usually a superset of C compiler options. The basic compiling information about C in section 16.2.3 is also applicable to C++, with the following exceptions[1]:

- Source filename extension conventions are compiler-dependent. Check the man pages for `CC` (upper case) to determine the extensions used on your system. Extensions include:

  `C` (upper case)

  `c` (lower case)

  `cxx`

  `cpp`

  `cc`

  `c++`

- The C++ compiler is invoked with `CC` (upper-case), `g++` or `gcc`
- There are additional compiler options specific to C++; see the man pages.

The C++ compiler may not yet be installed on your Fermilab UNIX system.

16.4 C and FORTRAN Compiler Options

The default compiler options will produce a usable program on any of the supported platforms, however they may not be optimal for many situations. Several additional options are discussed in this section.

A caveat: One very annoying "feature" of the `f77` and `cc` commands is that some of the options must be specified with whitespace (at least one blank character) between the option identifier and the option value, others without whitespace and still others with or without, according to the user's choice. For instance, on IRIX systems, there must be whitespace between `-o` and the name of the executable file but there must not be any whitespace between `-l` and the library file name.

If compilers are upgraded it is possible that some of these options could change. For full details on all the options, see the man pages and the vendor compiler documentation.

16.4.1 Commonly-Used Options

`-c`	suppress linking, produce object file `*.o`. The linker is called as part of the compilation process by default.
`-L` ***directory***	add ***directory*** to the default linker search list; not needed for user libraries. The `-L` option adds directories to the linker's default search path. `-L` directories are searched ahead of system library areas. A user library could accidentally match the name of an obscure system library, with startling results.

1. This doesn't apply to Gnu C++. The compiler command for both Gnu C and Gnu C++ is `gcc`, and the available Gnu compiler options are different from the vendor compiler options.

`-l`*file*	search library `lib`*file*`.a`, from the default areas (the `lib` gets prefixed and the `.a` appended automatically)

With user-written libraries, specify the libraries on the command line with their true file names and a full path, without using the `-L` and `-l` options.

`-o` *file*	(lower case `o`) produce executable program *file* rather than `a.out`
`-O`*n*	(upper case `O`) optimize at level *n* where *n* is 0, 1, 2, 3, or 4. The meanings of the different optimization levels vary from system to system. See the man pages for details.
`-w`	suppress informational and non-fatal compiler warnings
`-P`	(upper case `P`) run **cpp** (C preprocessor) only to produce `*.i` source listing
`-p`	(lower case `p`) enable profiling; see the man pages for details

Other commonly-used C compiler options are:

`-I` *directory_name*	extend include path
`-D`	set value of preprocessor macro

Other commonly-used FORTRAN compiler options are:

`-u`	IMPLICIT NONE
`-C`	(upper case `C`) check runtime subscript range

16.4.2 Recommended Options for General Use

As mentioned earlier, the default options may not be optimal for a number of very common situations, namely for debugging, moving binary code between non-identical machines, or tuning for best performance. Even for general usage, some non-default options have proven helpful in avoiding internal compiler limits and providing better compatibility for migrated code.

This list shows options by platform which apply to all situations, and which we recommend for general use. They can be used in addition to other options you might choose based on your specific needs. As stated above, if compilers are upgraded it is possible that some of these options could change, so you should always consult the man pages.

AIX	`-qextname -qrndsngl`
	`-qfltrap=invalid:overflow:zerodivide:enable`
IRIX	`-trapuv -lfpe`
OSF1	(none)
SunOS	`-xl -fnonstd`

Discussion of General Options

`-qextname`	(AIX) append underscore (_) to external names (default at Fermilab). This is needed if you use any Fermilab or CERN libraries. Better to just make it a habit.
`-qrndsngl`	(AIX) standard IEEE math on intermediate `REAL*4` results. This produces bitwise identical results to IRIX/OSF1/SunOS for most programs, making port testing easier.

`-qflttrap` (AIX) detect and report floating point errors. Without this, errors occur silently, and incorrect results may be produced. Unlike other platforms, these would not be NaN[1], but legitimate floating numbers. Do not use the `:imprecise` option here, as it appears to disable error detection entirely (AIX 3, XLF 2).

`-trapuv` (IRIX) set uninitialized variables to NaN, to help catch nonportable code and latent bugs at runtime.

`-lfpe` (IRIX) use the Floating Point Exception library, to report or core dump on errors. Without `-lfpe`, errors produce NaN values silently. Must be combined with the *TRAP_FPE* environment variable, or calls to `handle_fpes`, to be effective. *TRAP_FPE* must be set to the value:

```
OVERFL=ABORT;DIVZERO=ABORT;INVALID=ABORT
```

See section 9.1 for setting environment variables.

`-xl` (SunOS) Extended Language, for Fermilab-required extensions (see section 16.2.2)

`-fnonstd` (SunOS) trap floating point errors.

16.4.3 Debugging Option

The `-O`'s here are all upper case, for optimization.

`-g` include full symbol table for debugger. This option interacts with the optimizer differently on each platform, so we provide some usage notes:

AIX	you should not use `-O` with `-g`
IRIX	`-g` forces optimization off; use `-g3` to permit optimization
OSF1	`-g` changes default optimization to `-O0` (dash, letter-O, zero)
	An additional OSF1 option, `-synchronous_exceptions`, puts traps after each floating point operation. This is very slow, but will precisely locate floating point errors. Use this only for code which is known to produce floating point exceptions.
SunOS	you should use `-O1` with `-g`

16.4.4 Portability Option for AIX

`-qnomaf` Turns off Multiply/Add instructions in order to enhance portability.

16.4.5 ABI Options Under IRIX 6

Under IRIX 6, there are three Application Binary Interface (ABI) options available. The selected option specifies at compile time which subset of the processor chip's capabilities is to be used. In previous releases of IRIX, and in all releases of the other supported UNIX flavors (as of this

1. NaN is the IEEE standard Not-a-Number bit pattern produced when a floating point operation generates invalid results.

writing), the user is not given a choice of ABI. Under IRIX 6, choosing an ABI is generally necessary if you're doing mixed-language programming or using any libraries other than those supplied by the vendor. The available ABIs and their associated command line options are:

O32	(`-32`)	conforms to the ABI used with all prior IRIX releases
N64	(`-64`)	puts the processor into full 64-bit mode
N32	(`-n32`)	leaves the processor in 32-bit mode but takes advantage of a number of newer features and generally produces more efficient code on the newer processors

This topic is discussed in the Web page *IRIX 6 Application Binary Interfaces*, available under *Physics Applications* on the CD home page (look under *Cern Library at Fermilab*).

16.4.6 Speed Optimization Options

Note that on most platforms a combination of options is required for best optimization. Recall that these options may be used in addition to general recommended and other options. The `-O`'s here are all upper case.

AIX	`-O2`
IRIX	`-O3 -mips2`
OSF1	`-O3 -feedback -tune host`
SunOS	`-O3 -cg92 -libmil`

Discussion of Speed Optimization Options

Platform	Options	Comment
IRIX	`-mips2`	Use full R4000 instruction set. This is important on R4000 and later systems.
OSF1	`-feedback`	Order routines for best cache performance, based on actual program execution statistics.
	`-tune host`	Use best optimization for given generation of chip. The code will run anywhere, but is tuned for local speed.
SunOs	`-cg92`	SuperSPARC V8 instruction set Sparcstation 2 and later. See the man fpversion document.
	`-libmil`	Hardware-specific inline math

16.4.7 Load Map Option

The load map option is actually a linker option. On AIX, IRIX and OSF1 this requires passing to the link phase the option that controls production of a load map.

Platform	Option	Output
AIX	**-Wl,-bmap:*file***	***file***
IRIX/OSF1	**-Wl,-m**	`stdout` (SysV style list of input/output)
IRIX/OSF1	**-Wl,-M**	`stdout` (BSD style primitive map)
SunOS	**-m**	`stdout`

Additionally, under AIX two related options are available:

-Wl,-bloadmap provides additional internal loader information

-Wl,-bxref provides additional cross reference information

16.4.8 Special FORTRAN Compiler Options

Source Code Listing Option

Each platform has an option that produces a source code listing. The file extensions for these listings are platform-dependent.

Platform	Option	Listing Extension
AIX	**-qsource**	`lst`
IRIX	**-listing**	`L`
OSF1	**-V**	`l`
SunOS	**-Xlist**	`lst`

Saving Local Variables

The FORTRAN 77 standard allows subprogram local variables to become undefined between calls, unless saved with a SAVE statement. Many UNIX **f77** compilers *require* the SAVE statements for retained local variables. SAVE'd variables are usually called *static*, and unSAVE'd variables are called *automatic*. For programs not yet properly equipped with SAVE statements, **f77** command line options are available as follows:

Platform	Static Option	Automatic Option
AIX	(default)	**-qnosave**
IRIX	**-static**	**-automatic** (default)
OSF1	**-static** (default)	**-automatic**
SunOS	(default)	**-stackvar**

C Preprocessor

The C preprocessor (**cpp**), a macro processor, is used in **f77** for conditional compilation, macro expansion and source inclusion. If you plan to run **cpp**, remember to double any backslash characters (\\) in your code to prevent their misinterpretation as **cpp** escapes. The option shown in parentheses is the default option for the corresponding platform:

IRIX	(`-cpp`) `-nocpp`
OSF1	`-cpp` (`-nocpp`)
SunOS	any input file name `*.F` is automatically preprocessed by **cpp**
AIX	(see below)

The C preprocessor cannot be invoked through the `f77` command on AIX systems. It can, however, be invoked under AIX as a separate command:

`% /lib/cpp -P -C` ***infile outfile***

See the man pages for **cpp** syntax and usage information.

Extend Search Path for INCLUDE

Use the `f77` command line option `-I` ***path*** to extend the search path for the INCLUDE statement on AIX/IRIX/OSF1 (not supported on SunOS), where ***path*** is the path to the directory where the include files are found.

Internal Compiler Limits

You may need to set one of these if you are compiling a very large single source file.

AIX	`-NQ20000` Larger internal compiler tables. This value should be big enough for almost all programs.
IRIX/OSF1	`-Olimit 1500` Allows somewhat larger routines to be optimized.

16.5 FORTRAN Programming

There is some additional information about using FORTRAN in the UNIX environment that you will find useful to know.

16.5.1 External Reference and Entry Point Names

In order to avoid conflicts with the C runtime library when FORTRAN and C programs are included in a single program, most UNIX **f77** compilers internally append an underscore to FORTRAN external references and entry point names. At Fermilab we have set up all **f77** compilers to do this by default.

16.5.2 Separate Compilation of FORTRAN Subprograms: fsplit

By default, most **f77** compilers pre-link all the source code being compiled, even when you specify the `-c` option. If you compile a library with a single `f77` statement, it will usually contain a single module, and be linked as a whole.

The **fsplit** utility identifies and extracts subprograms from the original FORTRAN source file into individual files in the current directory. These files can then be compiled separately so that they retain their identity when assembled into a library.

The names of the extracted individual files are taken from their corresponding subprogram names. On some systems **fsplit** will overwrite any pre-existing file, including the original source file, whose name matches any of the subprogram names.

See the man pages for more information on **fsplit**.

16.5.3 AIX-Specific Issues

You must `CALL EXIT` to avoid having a `STOP` message printed on `stdout` at program termination.

The `OPEN` statement parameter `READONLY` is unavailable. All files are opened with `WRITE` access. This makes file sharing problematic under AIX.

16.5.4 Loading Block Data Modules

Many UNIX **f77** compilers enforce the standard restriction that variables in COMMON must be initialized only in BLOCKDATA subprograms.

To ensure the loading of BLOCKDATA subprograms from libraries, declare the BLOCKDATA program name as EXTERNAL in some important module which you know will be loaded.

16.5.5 Program Control

Command Line Arguments

A FORTRAN program can easily evaluate arguments included on the command line that runs the program. A couple of examples follow.

- The IARGC function returns the number of command line arguments:

  `N = IARGC ()` Sets `N` to the number of command line arguments

- The GETARG subprogram returns the value of a specified argument:

  `CALL GETARG(I , STR)` Puts the `I`'th argument into string STR

Environment Variables

The GETENV subprogram provides the values of environment variables. For example, to copy the value of variable `MY_OUT` into string `OUTFILE`, include in your source file:

```
CALL GETENV ( 'MY_OUT' , OUTFILE )
```

Printing

The usual FORTRAN carriage control characters placed in the first column of formatted output files are not interpreted by most UNIX text handling utilities. Use the UNIX **asa** utility to convert such FORTRAN output files to an equivalent standard ASCII text form. **asa** handles blanks, 0, 1 and + in column 1, removing any other characters. See the man pages for **asa** for details.

16.5.6 Future FORTRAN Enhancements

AIX XLF 3

This document describes the AIX XLF 2 compiler. IBM's standard **f77** compiler, XLF 3, is not yet available at Fermilab.

FORTRAN 90

The FORTRAN 90 standard includes FORTRAN 77 as a subset, and makes standard many of the extensions in common use. FORTRAN 90 is not yet commonly installed at Fermilab, and in fact we recommend that you avoid using FORTRAN 90 extensions until it is widely available. This document is written for **f77** users.

16.6 Obsolete Programming Features

You may encounter these features in older code.

AIX XLF

Some years ago the AIX FORTRAN compiler command was **`xlf`**. The standard **`f77`** command is now available under AIX, and should always be used in place of **`xlf`**. However, as mentioned earlier, you still need to type **`man xlf`** to access the man pages.

Calling BLOCKDATA

Some systems (VS-FORTRAN) required an explicit call to each `BLOCKDATA` routine if you wished to force loading of that routine from a library. This is not necessary on any supported UNIX system.

BUFIO

The Fermilab **bufio** product for accessing raw variable length records on tape and disk is no longer supported. Improved capabilities are being supported in **RBIO** and **DAFT** (see sections 15.2 and 15.3).

RANLIB (SunOS/4)

The RANLIB utility added necessary library symbol tables under SunOS/4. This is done automatically under SunOS 5.

ar -s Option (ULTRIX)

The `ar -s` option added necessary library symbol tables under Digital's ULTRIX[1] and some earlier operating systems. This is done automatically under all Fermilab supported systems.

16.7 C and FORTRAN I/O

This section mainly applies to FORTRAN. For C, all you need to know is that the **RBIO** and **DAFT** libraries mentioned below are available.

Recall that file names in UNIX are case sensitive. It is customary to use lower case for normal files, reserving upper case names like README for documentation and control files.

Note that you cannot use the shell metacharacter tilde (~) to specify a home directory within a C or FORTRAN program; `~` is valid only on a UNIX shell command line (for all shells but **sh**). `logdir` can be used within programs for this purpose (see section 6.1.2).

16.7.1 Records

The UNIX operating system treats a disk file as a sequence of bytes. Interpretation of data as records is entirely up to individual applications. The FORTRAN I/O libraries provide the necessary record handling for FORTRAN programs. READ statements return only the content of the records, and not the control words mentioned below.

Formatted records are terminated by a new-line character <Ctrl-l> (lower case L), consistent with other UNIX text handling programs.

Unformatted records are both preceded and followed by a 32 bit integer containing an exclusive byte count.

16.7.2 Tapes

Tapes of course have real physical records, and must be handled differently than disk files. A tape file is sometimes called 'character special' to indicate that it is not accessed on a character-by-character basis. Tape handling is covered in Chapter 15.

There are no industry-wide standard tape I/O routines callable from FORTRAN. Fermilab makes use of several locally-written packages, including:

RBIO	labeled and directoried tape access (see document SU0033)
DAFT	raw binary tape access (see document PN0504)

You can find C and FORTRAN examples in Appendix B of the *DAFT User's Guide*, document PN0504.

1. ULTRIX has been superseded by Digital UNIX (`uname` returns `OSF1`).

16.7.3 Standard Input and Output

In conformance with the FORTRAN standard, `READ`s and `WRITE`s to unit `*` are directed to `stdin` and `stdout`. You can `READ` and `WRITE` to units 5, 6, and 0 without an `OPEN`. They are preconnected to `stdin`, `stdout`, and `stderr`, respectively. If you `OPEN` and write to any other unit number `#` without specifying a file name, a default name of `fort.#` will be used.

16.8 Performance Tuning for C and FORTRAN

16.8.1 Optimization

Using the compiler `-O` (upper case) options can improve program execution speed by factors of 3 or more, depending on the application, over unoptimized code. Note that your libraries must also be compiled at the same level in order for this to be effective.

Beware that there are some optimizer bugs. You should always do a limited run initially with and without optimizer options, and check your answers.

For production running, use the appropriate hardware-specific optimizations for the systems running the code. These options typically tune for cache sizes, instruction sets, and other internal hardware features, resulting in sizeable speed gains. On some systems this produces an executable that will run only on the targeted architecture.

It is common practice to retain debugger symbol tables in production programs, with only a small speed penalty. You may have to exercise care that the `-g` option does not also disable optimization of such production programs. Under IRIX, you must use `-g3` to get both optimization and symbol tables.

See the suggested speed optimization options, and vendor documents for details.

Floating Point Errors

You can obtain substantial speed increases on some systems, especially AIX, by disabling the detection and trapping of floating point errors such as overflow, division by zero, and invalid values.

On the systems with the biggest gains, this practice can produce apparently normal, but incorrect, results. For example, 1000./0. can produce the result 1000. It is hardly necessary to point out that this sort of thing can produce surprising physics results! For this reason our recommended options for general use are set to at least detect and report floating point errors.

Qualifiers which force precise trapping of floating point errors are generally only used when tracking down known problems, as they can impose a large performance penalty.

16.8.2 Word Length

It may be tempting to use arrays of short words to 'save memory'. On previous generations of computers this could also speed execution. On RISC systems there is a big performance penalty for this practice.

The current generation of RISC processors are optimized for 32 and 64 bit operations. Operations on 8 bit or 16 bit words are performed several times more slowly. The processor must extract the necessary data into a longer word, perform the operation, and mask the result back into the original location.

Alignment of variables is important for the same reason. A misaligned 32 bit word requires even more shifting and masking than a 16 bit word, with an even greater performance penalty. If you must combine different length variables in a data structure such as a COMMON, place longer words earlier in the data structure.

16.8.3 Feedback

The speed of a program can be limited as much by memory access as by processor speed. Effective use of memory cache is critical to getting good performance.

Cache usage can depend on the details of the linking process. Arbitrary changes in the ordering of modules in the executable can result in nearly 20% differences in execution speed, for typical physics code. Small changes like switching between static and shared libraries, or modifying a single subroutine call in your code, can result in substantial changes in linking order and hence in performance.

For this reason, some vendors provide mechanisms for setting optimized module ordering in the executable, based on data from a trial run of the application. Under OSF1, use the compiler `-feedback` option; see the **f77** or **cc** compiler man page for details and examples.

16.8.4 Inlining

Many compilers provide options for replacing calls to external modules with equivalent inline code, to permit better optimization and reduce subroutine call overheads.

Physics code does not generally benefit measurably from such inlining. Inlining within a library makes the inlined modules nonreplaceable at link time, leading to confusing results and difficult debugging. In our recommended speed optimization options we stop short of the levels that introduce inlining.

16.9 C and FORTRAN Mixed Programming

It is possible, with a little care, to combine C and FORTRAN modules in the same program. Some of the issues that need attention include:

- variable types
- array indexing
- external names
- arguments
- commons
- I/O
- linking

For newly written C programs, you may wish to use the `cfortran.h` header file available in the **cern** product.

The July/Sept 1994 *CERN Computer Newsletter* (217) contains an extensive tutorial by Alfred Nathaniel on C and FORTRAN interfacing, available locally under the heading **Software Development** from the *UNIX Resources* Web page.

If you're programming under IRIX 6, you will need to choose an ABI. Refer to section 16.4.5.

We give here a summary of the techniques used on the Fermilab UNIX systems. In Appendix I you can find an example that illustrates much of the information in this section.

16.9.1 Variable Types

Generally, these variable types are equivalent:

FORTRAN	C
INTEGER*1	char
INTEGER*2	short
INTEGER*4	int
REAL	float
REAL*8	double
LOGICAL	(unavailable)

C strings are zero-terminated, and have no intrinsic length. FORTRAN character variable lengths are given by an internal descriptor. FORTRAN character variables passed to C routines should be copied and zero-terminated before they are used.

The internal representation of FORTRAN LOGICAL variables is usually non-0/0 for .TRUE./.FALSE. respectively, but it is best not to count on this.

16.9.2 Array Indexing

By default C starts indexes at 0 and FORTRAN starts them at 1. C and FORTRAN multiple index ordering is reversed. FORTRAN substring selection appears as the first C string index. See the following equivalence table:

FORTRAN	C
`intv(j)`	`intv[j-1]`
`intv(j,k)`	`intv[k-1][j-1]`
`char(j)(k:k)`	`char[k-1][j-1]`

16.9.3 External Names

By default on Fermilab UNIX systems, the **f77** compiler modifies FORTRAN subprogram and other external names. It forces each name to lower case, and appends an underscore. Thus FORTRAN label `SUBPROG` would become C label `subprog_`. For AIX, make sure the **`-qextname`** option is set (see section 16.4.2).

16.9.4 Arguments

FORTRAN subprogram arguments are always passed as addresses (C pointers). C programs can specify arguments as either pointers or values. FORTRAN CHARACTER arguments are passed as pointers, followed by a set of additional values (not pointers) at the end of the argument list, giving the length of each CHARACTER argument.

C routines can always call FORTRAN routines, with due attention being given to arguments. FORTRAN routines cannot call arbitrary C routines.

16.9.5 Commons

FORTRAN COMMON's are accessible in C as *extern structs*, with the same name mapping as is used for entry points.

FORTRAN	C
`COMMON /FOO/ I`	`extern struct { int i ; } foo_ ;`
`K = I`	`k = foo.i ;`

It is best to keep your FORTRAN COMMON variables aligned on natural boundaries[1], in order to avoid potential padding words which may be inserted differently by various FORTRAN and C compilers. You get natural alignment easily by putting longer variables before shorter variables in the COMMON.

16.9.6 I/O

Mixed C/FORTRAN I/O to the same file is not advisable. Mixed C/FORTRAN I/O to `stdout`, where `stdout` is the terminal, will usually work reasonably well, making debugging easier.

16.9.7 Linking

The easiest and safest way to link C/FORTRAN programs is to use the **`f77`** command, which automatically includes both C and FORTRAN run time libraries. If you insist on linking with the **`cc`** or **`ld`** commands, remember to add the options:

```
-lF77 -lI77 -lm
```

16.10 Executing a Program

Once you create an executable, you run it the way you do a normal UNIX command, that is by typing its name followed by appropriate options or parameters.

---

1. Keep all ***n***-byte variables' addresses an exact multiple of ***n,*** for example 0, 4, 8,... for a 4-byte quantity.

You must be aware that if you have not included "dot" (`.`) in your path, whenever your executable is in a directory not explicitly included in your path, you will need to prefix the executable name with `./` to run it. This was also mentioned in section 9.2 under *PATH*.

16.11 Debugging

16.11.1 FORTRAN Source Code Analyzer: FLINT

FLINT is a FORTRAN source code analyzer, a proprietary product of IPT. At Fermilab, **FLINT** is available for SunOS 5 and IRIX 5. The **FLINT** User Manual for UNIX is available on the Web (document number PU0099) and on-line in the file `$FLINT_DIR/MANUAL` wherever **FLINT** is installed and setup.

FLINT provides several analysis features:

- local and global analysis
- cross reference tables
- program statistics
- calling trees

FLINT reports the following error conditions:

- inappropriate arguments passed to functions or subroutines
- inappropriate library calls
- inconsistencies in common-block declarations
- non-portable code
- type usage conflicts across different modules
- unused functions, subroutines and variables
- variables which are referenced but not set

FLINT Options and Files

Use the **`-S`** (upper case) option to specify an output file name. The default extensions listed below are used. A separate file will be created for each component of the listing according to the additional option shown:

`lnt`	Analysis output
`xrf`	Xref table (**`-x`**)
`tre`	Call tree (**`-t`**)
`stt`	Statistics (**`-s`**)

Use the **`-L`** option to create a `*.lbt` **FLINT** library, containing all subroutine interface information needed for subsequent call usage tests. This library may be specified as **FLINT** input in place of the original source.

Use the **`-B`** option to create a `*.fdb` database file, containing all the symbol information needed for subsequent cross reference and call tree analysis. This database file may be specified as **FLINT** input in place of the original source.

Lowercase options may be combined. Use a double dash to disable an option (e.g., **`--w`**); this is useful for disabling a default option.

Uppercase options take parameters (with or without a space between the option and the parameter, depending on the option) and do not combine.

When original source is not available, `*.lsh` ASCII files may be created to describe calling sequences. The file `$FLINT_DIR/unixlib.lsh` describes the standard UNIX system calls.

FLINT Usage Examples

To see the standard analysis warnings for Isajet, enter:

```
% flint $ISAJET_DIR/isajet.f
```

To see a calling tree of the Isajet program, without warnings, enter:

```
% flint -t -Tcondensed --w $ISAJET_DIR/isajet.f
```

To see an options summary, enter:

```
% flint "-?"
```

16.11.2 dbx

dbx is a utility for source-level debugging and execution of programs written in C, C++, and FORTRAN. **dbx** allows you to trace the execution of a specified object file. You can step through a program on a line-by-line basis while you examine the state of the execution environment.

Programs compiled with the **-g** option of **cc** (and other compilers) produce an object file. This object file contains symbol table information, including the names of all source files that the compiler translated to create the object file.

dbx also allows you to examine *core files* via its **where** command. A core file contains the core image produced when the object file was executed, providing information about the state of the program and the system when the failure occurred. A core file named `core` is produced by default.

dbx commands can be stored in a start-up `.dbxinit` file that resides in the current directory or in your home directory. **dbx** executes these commands just before reading the symbol table.

There are some UNIX tools which provide a more sophisticated interface to **dbx**. See your local system documentation for information on GUI-based **dbx** tools. The product **ddd** (originally an interface for **gdb**) works as a front end for **dbx** in its more recent releases. It is currently available at Fermilab as part of the **gcc** product, but we expect to release it as a separate product soon. See the *DDD User's Guide* on the Web, document number DS0230.

Running dbx

To invoke **dbx**, enter the following command:

```
% dbx [options] [object_file [corefile]]
```

where ***object_file*** is the name of the file you want to debug.

Once **dbx** is running, you should see the **(dbx)** prompt. At this point you can start issuing **dbx** commands.

Commands

There are many **dbx** commands, all described in the man pages. Some of the basic commands are **run**, **where**, **print**, **stop**, **list**, **cont**, and **quit**:

`run [`*`arguments`*`]`	Begin executing the object file, passing optional command-line ***arguments***. The arguments can include input or output redirection to a named file.
`where [`*`n`*`]`	List all active functions on the stack, or only the top ***n***.
`print [`*`expressions`*`]`	Print the values of one or more comma-separated ***expressions***. To print values of two-dimensional FORTRAN array elements use the format: `print` *`array_name`*`[1,2]`
`stop` *`restriction`* `[if` *`cond`*`]`	Stop execution if specified ***restriction*** is true. Restrictions include (this is a partial list): `at` (source line) ***n*** `if` ***cond*** `in` (procedure or function) ***func*** ***cond*** (condition) is a Boolean expression; if it evaluates to true, then execution is stopped.
`list [`*`n1`* `[,`*`n2`*`]]` or `list` *`func`*	List the source text between lines ***n1*** and ***n2***, or on lines surrounding the first statement of ***func***. With no arguments, list the next ten lines.
`cont [at` *`n`*`] [sig` *`signal`*`]`	Continue execution from the point at which it was stopped if no arguments. Resume from source line ***n*** or, if a ***signal*** name or number is specified, resume process as if it had received the signal.
`status [>` *`file`*`]`	Show active `trace`, `stop`, and `when` commands
`delete [`*`n`*`]`	Remove traces, stops, and whens corresponding to each command number ***n***, given by `status`. If ***n*** is `all` remove all.
`quit`	Exit **dbx**.

Example

You may want to start by using **dbx** to set some break points within your code. To step through your code at the very beginning, you need to stop in the `MAIN` routine if you are debugging an object file created from FORTRAN source code (stop in `main` if your source is in C language). For example, you would type:

```
(dbx) stop in MAIN
```

Now you can issue the `run` command to start execution of your object file. You will get the process id and the name of the object file being executed.

```
(dbx) run
```

At this point, you may use the `list` command for the first 10-line listing of the source code:

```
(dbx) list
```

Use the `stop` command to set break-points at various lines or procedures within the object-file:

```
(dbx) stop at 10
```

```
(dbx) stop in sub123
```

```
(dbx) stop in sub456 if i == 24
```

The execution will stop in the example above at line 10, or in subroutine `sub123`, or in subroutine `sub456` when `i` is equal to `24`. To continue execution at any point in your debugging, issue the **`cont`** command:

```
(dbx) cont
```

To restart your debugging session, issue the **`rerun`** command:

```
(dbx) rerun
```

To exit **dbx**, type **`quit`**:

```
(dbx) quit
```

Usage Note

A user reports that when using **`dbx`** ***`object_file`*** **`core`** he has found it useful to turn on all but one of the IEEE arithmetic traps, in order to stop execution when the arithmetic fault occurs (instead of continuing with some default action and then reporting that the following IEEE arithmetic flags had been set). He located a spurious division by zero in this manner. The **`f77`** man page for Solaris describes the necessary flag value on the **`f77`** command line: **`-ftrap=%all,no%inexact`**. We have not researched this for the other UNIX flavors.

16.11.3 gdb

gdb, a GNU product, can do four general types of things to help you debug your programs:

- Start your program, and indicate anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened when your program stops.
- Modify your program, allowing you to experiment with correcting one bug and go on to find another.

You can use **gdb** to debug programs written in C or C++.

You can also debug programs written in FORTRAN, although **gdb** does not yet support entering expressions, printing values, or similar features using FORTRAN syntax. Furthermore, it may be necessary to refer to some variables with a trailing underscore.

See the document *Debugging with GDB*, document number PU0172.

16.11.4 purify

purify is a commercial product that detects memory corruption and finds memory leaks in your executable programs. **purify** is currently available on FNALU for Solaris. The command to run it is **`purify`**.

See the man pages for information on its syntax, options and uses.

16.11.5 CASEVision

CASEVision is an advanced debugging environment for SGI. It is currently available on both of the SGI interactive nodes on FNALU. The command to run CASEVision is `cvd`.

See the man pages for more information. Additionally, documentation is available through the SGI on-line documentation tool **insight** (see section 3.1.3).

Chapter 17: The make Utility

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. Although **make** can be used in a wide variety of applications, in this chapter we concentrate on its use in the area of software development. We describe how to define relationships between source, object, library and executable files for use by **make**, and how to invoke **make** in its simplest and slightly more complex forms.

17.1 An Overview of the make Utility

make is a command execution utility. You can use it to essentially automate any task in which one or more "target" file(s) requires updating via a shell command when changes have been made to any of its "required" files (files from which the targets are built). There is some preparation to do, but once that is complete, all you do is enter the `make` command!

make compares the modification dates of target files to those of their required files. For each file that needs updating, **make** issues the predefined update command(s) for the file. For example, if file A is a required file of file B (the target), and if file A has a more recent "last modified date" than file B, then **make** *re-makes* file B by issuing the specified update command(s). Target files that are found to be more recent than all their required files are skipped over.

make is especially useful in long-term software development projects that involve large numbers of source files, libraries, and executables connected by a complex set of relationships. You can use it with any programming language whose compiler can be run with a shell command.

make obtains information about the files, their relationships, and the specific update commands from one or both of the following:

- a specially formatted, user-supplied control file called the *Makefile* (see section 17.2)
- **make**'s set of built-in default rules (see section 17.6)

In preparing to use **make**, you generally need to write a Makefile. The Makefile defines the relationships among the constituent and target files of your project by listing the required files for each target file, and stating the shell commands that must operate on the required files to create or update the target(s). **make** implicitly treats all required files as targets, in an iterative manner. A file listed as a required file in one definition statement may also explicitly appear as a target in another statement. For example, in a program, typically the executable file (the final target) is created from object files, which are in turn created by compiling source files. Once you have a working Makefile, you just run the **make** program to perform all your updating tasks.

The **make** man pages give a full description of the command, its options and features, as well as the format and usage of the Makefile.

We have included two examples in Appendix I that illustrate many of the points discussed in this chapter.

17.2 The Makefile and its Components

The Makefile is a blueprint that you design and **make** uses to create or update one or more target files based on the most recent modify dates of the required files. The **make** command line syntax remains quite simple in the case where a Makefile is used:

% make [-f *makefile_name*] [*other options*] [*targets*]

If the **-f** option is not used, **make** checks for a Makefile of a particular name: it first looks in the current directory for a file named `makefile`, then for `Makefile`. If the Makefile is still not found, **make** looks in a couple of other places (see the man pages). In order to keep things simple and standard, we recommend that you always use the filename `Makefile` for your Makefile(s). Following this convention, you'll have only one Makefile in a directory. Your Makefile can contain instructions for building many different targets. When executing **make**, you can specify the desired target(s) on the **make** command line.

We can categorize the types of statements that can go in a Makefile as follows:

macro definition	A macro is a name that you define to represent a variable that may occur several times within the Makefile.
target definition	A target definition lists the target file, its required files, and the commands to execute on the required files in order to produce the target. (You can opt to specify the totality of this information in separate target definitions.)
suffix rules	Suffix rules indicate the relationship between target and source file suffixes (filename extensions). For example in FORTRAN, object files (`*.o`) are created from source files with a suffix of `f` (i.e. `*.f`). If no source file is explicitly given on a target definition line, **make** uses suffix rules to determine what source file to use to produce the target.
suffix declarations	Suffix declarations are lists of suffixes (file extensions) used in suffix rules.

Each new line in the Makefile starts a new definition, except that in target definitions:

- shell commands with leading tabs are part of the previous definition (a standard Makefile format that you need to recognize, but that we suggest you avoid using for reasons explained in section 17.2.2.)
- shell commands can be continued in standard UNIX format, using a trailing backslash (\)

Blank lines are permitted between definitions. Comments can be included after a pound sign (#). To use a literal pound sign, precede it with a backslash, i.e. `\#`.

17.2.1 Macros

A macro is a name that you define to represent a variable that may occur several times within the Makefile, or that needs to be updated frequently. Macros make maintenance of your Makefile much easier. They are commonly used to define settings, platform-specific commands, lists of required files for a target, lists of command options, and so on. **make** reads all the macro definitions before executing any commands. It is often convenient to put all the macro definitions at the head of the Makefile, but this is not necessary.

Format and Usage

Macro definitions are of the form:

macro_name = value

where ***macro_name*** is the name you want to use in place of the longer ***value.*** Everywhere in the Makefile that ***macro_name*** is found, **make** substitutes ***value***. For portability, if ***value*** contains any blank spaces that it is supposed to have, the ***value*** string must be enclosed in quotes in the definition statement.[1] Backslashes can be used to continue the same line; you cannot put a new line in a macro value.

Once a macro is defined, you refer to its value in the form `$(`***macro_name***`)`. If ***macro_name*** is a single character, you can omit the parentheses.

As an example, let's define a macro FFLAGS to set some FORTRAN default options to use with the **`f77`** command:

```
FFLAGS = "-O2  -w -Olimit 1500 -nocpp "
```

You can now refer to the value of FFLAGS within the Makefile in the form `$(FFLAGS)`. For example, you might include a target definition line like the following (the format is explained below in section 17.2.2):

```
foo : foo.f ; f77 -o foo $(FFLAGS) foo.f
```

Special Macros

To be sure that **make** uses the standard, portable Bourne shell, always include in your Makefile a macro of the form:

```
SHELL = /bin/sh
```

Some versions of **make** default to your current interactive shell if you don't include this explicit SHELL macro in your Makefile. The standard Makefile format that we describe in this chapter assumes **sh** as the command interpreter.

Macro Sources

Macro definitions are similar to and can take default values from environment variables. **make** gets additional macro definitions from the following sources, and applies them in the order shown:

1) currently defined environment variables

2) built-in **make** rules

3) definitions in the Makefile

4) definitions on the command line

In other words, this list is in order of reverse precedence; a value from a source later in the list overrides a value applied from an earlier one. Using the **`-e`** option on the **`make`** command line swaps the first two; see section 17.3.1.

---

1. Be aware that this quoted value is what **make** hands to the shell. The shell then hands the macro, e.g., `$(FFLAGS)` to a program to execute. If you want leading or trailing spaces in ***value*** to be included in the command, specify the macro in double quotes in the command statement, e.g., `"$(FFLAGS)"`.

Symbols Used in Macros

`$$`	maps to a literal dollar sign
`$*`	is used in suffix rules (see section 17.2.3); it refers to the filename without the suffix
`$@`	is the current target make is processing
`$<`	is the implied source in a suffix rule

17.2.2 Targets

Targets are the files that you want to update or create. A complete target definition includes the name of the target, the files needed to build it (its required files), and the commands that must be executed to recreate the target.

Format

The standard Makefile format using Bourne shell conventions calls for:

- a space between listed targets
- a colon (`:`) between the last target and the first required file
- a space between listed required files
- a semi-colon (`;`) after the last required file if commands follow on the same line
- a semi-colon (`;`) between the successive listed commands

A simple target definition with a single target, required file, and command can be written in the form:

target **:** ***required_file*** **;** ***shell_command***

or, using a more traditional format, listing the command on the next line (note that the semicolon (`;`) is omitted here):

target **:** ***required_file***

`{tab}`*shell_command*

In this traditional Makefile format, the commands beyond the first line of a definition must have leading tabs, and there must be no intervening blank lines.

There are two reasons to avoid this second format: first, when working with the Makefile it is hard to see the difference between a tab and blanks, and secondly, some editors change tabs to blanks (or vice versa), which causes problems (neither **emacs** in default mode nor **vi** changes tabs to blanks). You can check your file to see if it contains tabs or blanks by running the command:

`cat -tev` ***filename***

We propose as an alternative that you use the backslash character (`\`) to continue the single definition line down as many physical lines as you have to go. Also, note that under AIX the standard "tabbed" format can be unpredictable; it is therefore safer for *several* reasons to use our suggested format on this platform.

This "safer" format which we suggest if the entire definition doesn't fit on the first line is:

target **:** ***required_file*** **;** **`\`**

shell_command

A target definition line can contain more than one of each element type. Or, it may contain only two of the three element types. A more complex definition in our suggested format looks like:

```
target_1 target_2 ... : required_file_1  required_file_2 ... required_file_n \
required_file_n+1 ...; \
shell_command_1 ; shell_command_2 ; shell_command_3 ; ... ; \
.
.
.
... ; shell_command_m
```

If there is more than one target (e.g. ***target_1 target_2***) in one definition, the commands are attempted separately for each target.

If you find it easier, you can list multiple required files for a single target in separate target statements. However only one statement for a given target can include commands, and therefore must include all the commands. Here is an example of this alternative format:

```
target : required_file_1
target : required_file_2
target : required_file_3
target : ; shell_command_1 ; shell_command_2 ; shell_command_3 ; ...
```

It can be confusing if you separate a series of statements like this from one another in the Makefile; if you use this format, keep the statements together!

Usage

The relative modification times of the ***target*** and the ***required_file(s)*** determine whether the listed commands will be executed. If the target file is found to be missing or to be older than any of its required files, **make** executes the commands to rebuild it. **make** treats the required files iteratively as targets, whether or not they are explicitly listed as targets in subsequent target definitions, and rebuilds them as necessary before rebuilding the final target.

17.2.3 Suffix Rules

A suffix is essentially a file extension. A suffix rule defines the relationship between target and required files by their file extensions. A suffix rule is much like a target definition except that it uses implied rather than explicit file names for target (output) and required (input) files.

A suffix rule is of the form:

.insuffix.outsuffix : ; command

The dots are really part of the suffixes themselves. As an example, assume you have a set of target files to rebuild whose filenames are all of the form `*.b`. The required files for these targets have filenames of the form `*.a`. Define the suffix rule:

`.a.b : ;` ***command(s)***

make expands this to the following target definition for all files ending in `.a` in the current directory:

filename*`.b` : *filename*`.a` ; *command(s)

If you define suffix rules specifically for intermediate files in a process, you still need to include a rule for the final and original files, for example, if you define:

```
.tex.dvi: ; latex $*  #latex cmd makes .dvi files from .tex files
.dvi.ps: ; dvips $*   #dvips cmd makes .ps files from .dvi files
```

You still need to provide the rule:

```
.tex.ps: ; latex $*; dvips $*  #make .ps files from .tex files
```

make is not sophisticated enough to do the transitive closure on suffix rules.

Note that suffixes don't have to start with a dot (`.`).

17.2.4 Suffix Declarations

Any suffix that you use in a suffix rule must be listed explicitly in a `.SUFFIXES` declaration in the same Makefile unless it is included in **make**'s built-in suffix declarations (see section 17.6). A suffix included in the built-ins can also be included in a suffix declaration in the Makefile.

Suffix declarations are really target definitions for a special target named `.SUFFIXES` and they contain no commands.[1] They are of the format:

`.SUFFIXES : .a .b`

where **`.a`** and **`.b`** are suffixes. This example suffix declaration would allow you to include the suffix rule from 17.2.3 in your Makefile:

`.a.b : ;` ***command(s)***

You may combine all the suffixes you use in the Makefile into one `.SUFFIXES` declaration, or group them into separate statements.

Suffixes that you declare add to the built-ins; they do not replace them.

17.2.5 Control Files within a Makefile

You may encounter situations where it is useful to pipe input or output of one command to another within a Makefile. You can echo a small control or data file within the Makefile, rather than maintaining a separate external file. In the following target definition example from an Isajet Makefile, several control statements are echoed into the **patchy** utility in order to extract `isaint.f` from the `isajet.car` patchy library:

```
isaint.f : isajet.car ; \
(echo "+USE,*ISAJET,$(MACHINE).              "; \
 echo "+USE,INTERACT.     INTERACTIVE PATCH  "; \
 echo "+EXE.                                 "; \
 echo "+PAM,T=C.                             "; \
 echo "+QUIT.                                ";)\
 | ypatchy isajet.car  isaint.f  \& genint.lis .GO
```

1. Most versions of **make** have other special targets (sometimes called magic targets) besides `.SUFFIXES`. These special target names always start with a dot (`.`).

17.3 Running make

17.3.1 General Usage

The **make** utility is invoked with the `make` command. The command syntax is:

```
% make [options] [targets]
```

Several ***options*** are available, and are described in the man pages for **make**. We provide a list of some of the commonly used options:

`-n`	Preview the commands, don't execute. Very useful for testing.
`-d`	Debug; list the operations used and why ("read **make**'s mind"). Available on all platforms except OSF1.
`-e`	Environment variables override built-ins.
`-f` ***makefile_name***	If your Makefile has a name other than `makefile` or `Makefile`, use this option followed by the file's name to identify it (leave a space between the option and the filename).
`-p -f /dev/null [|less]`	Print the built-in rules (see section 17.6).

The ***targets***, as mentioned earlier, are the files that you want to create or update. **make** searches the Makefile to find a target definition statement for each target listed on the command line.

You can include macro definitions on the command line which get applied after the assignments made in the Makefile. For example:

```
% make "CC = gcc" target
```

17.3.2 Usage without Specifying Target

When **make** is invoked without a specified target, the first non-suffix target in the Makefile is used. The command is simply:

```
% make [options]
```

For larger products, it is standard practice to name this target *all* in the Makefile, and in the list of required files to list the individual targets which together actually produce the full product. For example, the first Isajet target definition is:

```
all : isadecay.dat \
      isatext.doc  \
      isajet.a     \
      isaint       \
      isasusy
```

Notice that here no commands are listed. They would appear in the subsequent target definitions.

17.3.3 Usage without a Makefile

The extensive built-in rules let you use **make** quite effectively without having your own Makefile. Section 17.6 provides a brief explanation of the built-in rules. **make** will look for any file whose name matches that specified on the command line and which has a file extension that identifies it as a reasonable source. For example, to produce the executable `foo` (the target) from a `foo.c` or `foo.f` source that exists in the current directory, enter:

```
% make foo
```

make will look for the C or FORTRAN file as the source file for this target. Taking the FORTRAN program and no options as an example, this command is equivalent to (see section 16.2.3):

```
% f77 -o foo  foo.f
```

17.4 "Housekeeping" Targets

It is common practice to have a "housekeeping" target which removes stray files from the working directories. Typically you would run **make** on this target after you've completed the **make** operation on your principle target(s). It is conventional to call this target *clean*. Here is an example which removes unnecessary generated files from several different directories. The target definition has no required files:

```
clean: ; \
  rm  -f *.bak    ;\
  rm  -f *.lis    ;\
  rm -rf Maketemp ;\
  cd example/isaplt ; rm -f *.lis*   ;\
  cd  ../jet        ; rm -f jet.log*
```

You need to determine what stray files will be generated in your case, and define your commands accordingly. Run **make** on the *clean* target by entering:

```
% make clean
```

You may wish to define different levels of housekeeping targets. One that clears out everything, leaving only the original files you had before running **make**, is often named *clobber*.

17.5 Portability

It is desirable for your Makefile to be portable across different UNIX platforms. Why might this be a problem to implement? As mentioned earlier, **make** does all macro processing before any commands are executed. Therefore environment variables set in shell scripts executed by **make** have no effect on **make**'s macro definitions. And standard **make** doesn't support conditional macro definitions. So, how can you write a portable Makefile?

A Solution

1) Create a script for setting environment variables.

2) Have **make** run this script ("source" it; see section 4.4) from within the Makefile.

3) After it runs the script, have **make** rerun itself with a different target and the original command line options. A *$(MAKE)* macro which causes **make** to rerun itself is a standard feature.

An example of this technique follows.

Example

Create a portable shell script (named, for example, `Makeenv`) which sets appropriate environment variables for the Makefile. Here is a simple `Makeenv` script which defines the macro `F77` based on the current platform as determined by the command **`uname -s`**:

```
export F77 MACHINE

case `uname -s` in
IRIX)
        F77="f77  -O2  -w -Olimit 1500 -nocpp "
;;
OSF1)
        F77="f77  -O1  -w -Olimit 1500 -nocpp -static "
;;
esac
```

The Makefile is below. Run **make** with the target `isaint`. The Makefile runs `Makeenv`, then **make** reruns itself with the target `do_isaint` and the correct `F77` value:

```
isaint    : isaint.f ;  . Makeenv; $(MAKE) do_isaint
do_isaint : isaint.f ; $(F77) isaint.f  -o isaint
```

Other Utilities

There are other utilities available for more complicated cases:

- **gmake** (Gnu make), part of the Fermilab **gtools** product, has some nice portability features, and supports other advanced features like parallel compilation on multiprocessor systems.
- There are preprocessors for building locally tailored Makefiles in very sophisticated ways, including **gnu configure**, **premake**, and **imake**.

17.6 make's Built-in Rules

make comes equipped with a long list of built-in defaults to make your job easier. You are free to override any of them in your Makefile. The defaults fall roughly into four categories:

1) macros that reflect your current environment variables

2) macros that define standard compilers and options

3) suffix rules for finding required files when building targets

4) a list of known suffixes

To get a listing of all the built-in macros and rules, enter the command:

```
% make -p -f /dev/null [| less]
```

Depending on your platform, there may be nearly a thousand lines of definitions, so you might want to pipe this to **less**, or redirect the output to a file.

17.7 A Few Caveats...

1) Recall that in the traditional Makefile target definition format successive commands are entered on successive lines, each starting with a tab. Be aware that each of these command lines runs in a different shell. Two important implications of this are:

   a) if you have issued a change directory (`cd`) command, it does not carry over to the following line(s)

   b) environment and shell variables do not carry over to the following line(s)

The format which uses a single logical line for the entire definition avoids this problem.

2) If you use non-shell commands (for example `ls`) in definition statements, be aware that the output may vary from platform to platform. For this reason it is best not to rely on the specific output format of these commands.

Chapter 18: Code Management

This chapter introduces the recommended code management solution for UNIX, **CVS** (Concurrent Versions System). We also introduce an alternative that is currently being used by a couple of Fermilab experiments, **UCM** (UNIX Code Management)[1]. Both packages use **RCS** (Revision Control System) as the underlying protocol. We provide basic information only, and refer you to the complete manuals for these utilities for detailed information.

RCS provides a version control system with which you can record the history of your source files. An **RCS** file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. Only the differences between versions are kept.

CVS and **UCM** implement the **RCS** features differently, but both assume availability of **RCS** commands. **CVS** allows concurrent development, whereas **UCM** was created to replace **CMS** (a VMS source code management system) and implements the same sequential development philosophy of that system. Both systems provide easy extraction of either a release version or the latest version, and they allow you to create tags for release versions of the software.

See *Code Management* under the **Software Development** heading on the *UNIX Resources* Web page for more information. Alternatively you can search the on-line product documentation database for **CVS**, product number PU0189, or **UCM**, product number PU0254.

18.1 CVS

The **CVS** product allows many programmers to work on the same code simultaneously, each in his or her own directory, and it merges the changes when they are finished. There is no built-in mechanism to prevent concurrent development.

CVS stores all files in a central repository, a directory populated with a hierarchy of files and directories. The files are organized in modules, where a module is made up of one or more files, and can include files from several directories. It is typical to define one module per project. Although the structure of the repository and modules file interact with your build system (e.g., Makefiles), they are essentially independent.

18.1.1 Accessing CVS and Obtaining the Manual

To set up the **CVS** product:

```
% setup cvs
```

1. **UCM** is currently supported only within the experiments using it; although it is provided as a **UPS** product, it is not centrally supported at Fermilab.

The **CVS** document (in postscript format) can now be found in the directory pointed to by `$CVS_DIR/doc`. You can also find documentation on the Web for **CVS**, product number PU0189.

CVS points to the editor defined in your *EDITOR* variable for entering log messages (see the **`cvs commit`** command below). If it is not set, the editor defaults to **vi**.

18.1.2 Basic CVS Commands

Normally you never access files in a repository directly; you use the **CVS** commands to get your own copy of files. The common commands and their functions are listed below (we do not provide information on options here; see the document referenced above):

`% cvs import`	installs new release of source in **CVS** repository the first time
`% cvs co` ***module***	creates new directory called ***module***, populates it with source files; this allows you to "checkout" your own working copy of the source ***module***
`% cvs checkout` ***module***	equivalent to **`cvs co`** ***module***
`% cvs commit`	commits changes you have made; opens editing session for entry of log message
`% cvs rtag` ***release_number module***	tags a release
`% cvs export -r` ***release_number module***	extracts the specified release without **CVS** administrative directories

18.2 UCM

UCM is a replacement for **CMS** which has been widely used at Fermilab on VMS systems. To enforce sequential development, it uses a reservation system to ensure that a second programmer cannot change a module that another programmer is currently editing. **RCS** (Revision Control System) is a readily available UNIX product which is very similar to **CMS** except that it operates on individual files rather than on libraries. The component of **UCM** which actually replaces **CMS** is called **UVM** (UNIX Version Management), and it is an interface to **RCS**. **UCM** also contains tools which extend its use beyond the scope of simple source code management.

18.2.1 Accessing UCM and Obtaining the Manual

To access **UCM**, you first need to set it up.[1] Include in your login files or enter:

```
% setup ucm
```

1. On some systems, you will have to setup **CVS** to access **RCS** before using **UCM**. This is not necessary where FUE is installed.

UCM, product number PU0254, is fully documented on the Web. Once the product is set up, you can also find the **UCM** manual in the directory pointed to by `$UCM_DIR/doc`. In addition to providing more in-depth information on the concepts and commands introduced here, it provides examples of updating include file directories, updating an object library, and maintaining release versions.

18.2.2 Basic UCM Commands

You use **UCM** by executing commands provided by two of its constituent utilities, **uvma** and **uvmi**. The **uvma** commands are used for creating and maintaining libraries, and the **uvmi** commands provide information to the user. These two command sets can be invoked directly through **`uvma`** and **`uvmi`**, or indirectly through the generic command **`uvm`**.

Typing **`uvm`**, **`uvma`**, or **`uvmi`** without any commands, arguments or options will display the list of commands available to the corresponding command set.

Some basic **uvma** commands are listed below with brief descriptions of their functions:

`% uvm create library` ***library***

creates a **UVM** source code library

`% uvm create element` ***library/file***

creates a **UVM** element in the library

`% uvm fetch` ***library/file*** creates a copy of the specified **UVM** element(s) in the user's local area

`% uvm reserve` ***library/file*** extracts a working version of the latest revision of the specified **UVM** element(s) into the user's local area and locks the RCS file[1]

`% uvm replace` ***library/file*** creates a new revision of the specified **UVM** element(s) from working file(s) in the user's local area

`% uvm tag` ***library/file*** **`-n`** ***tagname***

assigns a symbolic tag name, ***tagname***, to the specified revision of the **UVM** element(s)

`% uvm group` ***library/file*** **`-g`** ***groupname***

adds a **UVM** element to a group specified by the **`-g`** option

1. Elements must be reserved before they can be changed.

Appendix A. VMS Migration for the Impatient

So, you've decided you're ready to convert (or you've run up against a deadline!), but you don't know the first thing about UNIX. Here's enough information to get you moved over. You can use the rest of the manual to learn about UNIX afterwards.

See the introductory remarks to Chapter 12 before continuing.

Read this entire appendix before running any of the shown commands!!

Keep these typeface conventions in mind as you read the commands:

`typewriter-bold`	In text, used to indicate commands and prompts. In command formats, indicates what the user types "as is".
bold-italic	In command formats, indicates variables for which the user must make context-specific substitutions.

A.1 The Two Necessary Commands

To convert to UNIX, you need to move your mail and your files over from your VMS node (***vmsnode***). To accomplish this while preserving the mail folder and directory structures you had set up on VMS, enter the following two commands exactly as shown from your new UNIX node. (In case you really haven't read any part of this manual yet, the `%` represents the UNIX prompt.)

For your mail, use the command:

```
% setup mh ; fvms2mh -vms vmsnode
```

Note that ***vmsnode*** must be an individual node, not a cluster alias. For your files, the command is:

```
% mkdir vms ; rcp -r vmsnode:'[...]*.*' vms
```

A.2 OK, What's the Catch?

You're right, there's a little more to it. You must have selected a UNIX host to which you will copy your mail and other files. On the FNALU cluster, we suggest that general users choose the FSUI02 node. You must have learned to log in to UNIX, edit and manage your files, and must have selected a mail reader (presumed to be **mh**, **exmh** or **pine**).

In addition:

- You must have **rcp** access from your UNIX to your VMS account. This takes two steps:

1 Create a `.RHOSTS` file in your VMS `SYS$LOGIN` directory containing one line for each UNIX host that needs automatic access to your VMS account. If your username is different on any machine, include a second line with your username. For example, the contents might look like:

```
fsgi02.fnal.gov
fsgi02 aheavey
fdei01.fnal.gov
fdei01 aheavey
```

2 Make sure your `SYS$LOGIN:LOGIN.COM` does not print anything out. To ensure this, include the following line at the top of the file:

```
IF ( F$MODE() .EQS. "OTHER" ) THEN EXIT !  'F$VERIFY(0)
```

If you can no longer log in to edit your files, send mail to *compdiv@fnal*, and someone can fix it for you.

- For the mail copy, be sure you are using **mailtools v2_2** or later, as installed on the FNALU nodes and most other public nodes at Fermilab. Earlier **mailtools** versions do not support the **`fvms2mh`** command name described here, and may have quota and cleanup problems. To see whether **mailtools v2_2** is current, type **`ups list -a mailtools`**. If it is not current, select it explicitly by typing **`setup mailtools v2_2`**.
- The utility **fvms2mh** brings your mail folders over in **MH** format. If you plan to use **pine** but want to leave your folders in **MH** format, **pine** must be configured to recognize them. To do this, go to **pine**'s SETUP CONFIGURATION menu. There are many options available. Your configuration list should include at least the following:

```
inbox-path           = #mh/inbox
user-domain          = fnal.gov
folder-collections   = mail/[]
                       #mh/[]
[X]  enable-aggregate-command-set
[X]  enable-alternate-editor-cmd
[X]  quell-user-lookup-in-passwd-file
```

- If you choose to use **pine** exclusively, you will want your folders in the **pine** format. First complete the configuration above (with the exception of `inbox-path`). You can move entire folders between the **pine** and **MH** collections as follows:

1 While in **pine**, go to the `FOLDER INDEX` screen for the desired folder.

2 Enter **`;`** (semi-colon) to issue the **pine** Select command.

3 Enter **`A`** to select All messages.

4 Enter **`A`** to Apply the command.

5 Enter **`S`** for Save.

6 Use <Ctrl-n> or <Ctrl-p> as necessary to choose the destination folder collection, then enter the new destination folder name. Or just type in a folder collection and name. Respond `Y` to create the new folder.

A.3 Whoa! Too Fast!

If converting in two easy steps doesn't suit your style, you can take somewhat smaller steps.

1) You should probably first forward your current new mail to UNIX, and get used to the **exmh** or **pine** mail readers.

2) Once you've stopped receiving VMS mail for a while, run the first command in section A.1. As a yet more conservative option, check out the "semi-automatic" mail conversion method in Appendix H.

3) Then, when you're happy with the UNIX copy of your mail, delete the original VMS mail files. You may wish to back them up or archive them before deleting.

4) Now to convert your files, run the second command in section A.1 *exactly as shown.* Changes to any of the wildcarding is likely to do nasty things (for instance, moving the oldest rather than most recent version of each file, or dumping everything into a single directory rather than keeping your directory hierarchy intact).

Appendix B. UNIX Product Support (UPS) Overview

In this appendix we discuss the Fermilab product support structure, **UPS**. We recommend that you read and understand this material before performing any of the tasks described in Chapter 10.

The information in this appendix has been taken from *UNIX Product Methodology at Fermilab: Guide to Using UPS v3 and UPD v2 for Product Maintenance, Installation, Distribution and Development* (GU0014), but covered here in much less detail, appropriate to the different audience. We refer you to that document if you need further information.

Notice of Upcoming Changes

UPS and **UPD** are currently undergoing redevelopment with a significantly different design. The new versions and accompanying documentation are due for release in the first half of 1998.

B.1 Introduction

The methodology and infrastructure for product support and distribution under FUE is provided via a software support toolkit called UNIX Product Support (**UPS**). **UPS** was developed with the goal of providing a uniform and consistent interface for the management, distribution, installation and use of all the UNIX software it makes available.

UPS consists of two parts:

- one or more databases
- a set of procedures/programs to manipulate database

A **UPS** database is a directory containing an ASCII file for each individual product. This file contains information about and pointers to all of the installed copies of the product. Product maintainers use **UPS** commands to add, delete and modify product information in these files (these activities are beyond the scope of this document). Products supported under this structure are called **UPS** products. A **UPS** product is comprised of one or more applications, each of which is invoked by its associated command or commands[1].

UNIX Product Distribution (**UPD**) is a companion product to **UPS**, and provides the functionality for distributing products, generally stored as tar files, from a remote host to a local system. A convenient menu interface is provided that includes the **UPD** and **UPS** functionality necessary to complete an entire process of listing available product instances on the distribution node, copying a product instance from the distribution node, installing it on the local node, and declaring it to the local **UPS** database. The **UPD** interface also can be used to list the files in or extract a file from a remote tar file.

1. Most applications have a single invoking command. However some applications have a set of commands, each of which invokes a different function. See section 10.2.4.

B.2 The UPS Environment

On a FUE-compliant system the standard default login files (also known as *Fermi files*[1]) set your environment to facilitate **UPS** operations, among other things. Regarding shells, **UPS** supports both UNIX shell families, C (Berkeley) and Bourne[2]. At login time, the environment variable *UPS_SHELL* gets set to indicate the family of the shell in use. Any shell within that family can then be used for **UPS** operations.

The Fermi files are designed for interactive use only and will break if they are called from a non-interactive script. You need to explicitly include the following actions in the order shown in any script that you will use to setup products (i.e. in which you use the **`setup`** command to access one or more products; **`setup`** is described in section 10.2.2):

For the C shell family:

```
source /usr/local/etc/setpath.csh
source /usr/local/etc/setups.csh
```

For the Bourne shell family:

```
. /usr/local/etc/setpath.sh
. /usr/local/etc/setups.sh
```

The `setpath` script sets a reasonable default starting path. The `setups` script (read "set **UPS**") sets the **UPS** environment variables described below[3].

The login files also set the alias for **`setup`**. **Note that if you change shells or processes, this alias is no longer available to you.** In this case you need to manually re-source the `setups` script in your new shell.

If you have made any changes to your login files, make sure that the following lines are still included before using **UPS**:

For the C shell family:

`.cshrc` includes	`source /usr/local/etc/fermi.cshrc`
`.login` includes	`source /usr/local/etc/fermi.login`

For the Bourne shell family:

`.profile` includes	`. /usr/local/etc/fermi.profile`
`.shrc` includes	`. /usr/local/etc/fermi.shrc`

Three environment variables get set by the setups script at login:

PRODUCTS	If only one **UPS** database is defined, this points to it; if two or more are defined, this variable can be set as the space-separated list of **UPS** databases. The order of the databases in this list reflects the order of precedence for accessing products.
UPS_DIR	This points to the top level directory (called the product root directory) of the current instance of **UPS** (read sections B.8 through B.10 -- they're short! -- to understand the term "current instance").
UPS_SHELL	As mentioned above, this indicates the shell family in use; it is set to either `sh` or `csh`.

---

1. See Chapter 9 and Appendix C for descriptions and listings of the Fermi files.
2. The C shell family includes **csh** and **tcsh**; the Bourne shell family includes **sh**, **ksh**, **bash** and **zsh**. **zsh** is not supported at Fermilab.
3. These scripts are included in the FUE package **systools**.

You can use the environment variable *PRODUCTS* to list all the available products. For example, if it is set to the following list of **UPS** databases:

```
/home/dcdsv0_lv0/aheavey/ups_play/ups_database/declared
/usr/products/ups_database/declared
/afs/fnal/products/ups_database/declared
```

then the command:

```
% ls $PRODUCTS
```

will yield information similar to the following (edited for brevity)[1]:

```
/afs/fnal/products/ups_database/declared:
acrobat          frame            lund             tcl
admintools       fslib            mad              tclx
afsemu           ftcl             mailtools        tex
...text removed...
flint            ktevana          softwindows      xpdf
fmb              lapack           stdhep           zephyr
fmss             libgpp.obs       systools

/home/dcdsv0_lv0/aheavey/ups_play/ups_database/declared:
cedit     exmh      fermitpu  glimpse   juke      www

/usr/products/ups_database/declared:
altuascvs    fulib      lund         qq           tk           www
chekker      funkern    mbone        req          tpu          xemacs
conv2html    futil      mgui         sdsscvs      uas_build    ximagetools
...text removed...
flint        html       ntp          tcl          upp
fmb          ispell     ocs          tclx         ups
fonts        juke       olscvs       teleserver   webservers
```

When you setup a product, the environment variable *{PRODUCT}_DIR* gets defined, where *{PRODUCT}* is the name of the product in upper case, for example *WWW_DIR*. This variable points to the root directory of the particular copy of the product that you setup; for example after running:

```
% setup www v2_7
```

the variable *WWW_DIR* is defined as something like `/{ups_dirs}/products/SunOS+5/www/v2_7`. You can then run a command like the following:

```
% ls -al $WWW_DIR
```

1. Notice that the `ls` command does not list the databases in the order of precedence according to the *PRODUCTS* variable, but rather, alphabetically.

which returns a list of the files and subdirectories in the product root directory for **www** version v2_7, for example:

```
total 40
drwxr-xr-x   7 products g150        2048 Jun 19 18:54 .
drwxrwxrwx   5 root     root        2048 Sep  9 23:16 ..
-rw-r--r--   1 products g150         165 Jun 18 11:31 BUILD_INFO
-rw-r--r--   1 products g150        4703 Oct 16  1995 Makefile
drwxr-xr-x   2 products g150        2048 Aug  9 18:19 bin
drwxr-xr-x   2 products g150        2048 May  5  1996 etc
drwxr-xr-x   2 products g150        2048 May  5  1996 include
drwxr-xr-x   4 products g150        2048 May  5  1996 lib
drwxr-xr-x   2 products g150        2048 May  5  1996 ups
```

B.3 UPS Products

Products distributed and managed by the **UPS** system on a distribution or user node are often called *UPS products*. **UPS** products can be maintained on different disks, and/or in different directories. To be included under the **UPS** umbrella on a node or cluster, products must be installed (preferably in an area specially designated for products, although **UPS** does not require this) and declared to a **UPS** database. Product declaration can be done in **UPD** (see section 10.3.1) or via the **UPS** command **`ups declare`** (not covered in this manual).

On the distribution node, all the constituent programs and/or data files of a particular copy of a **UPS** product (collectively called an *instance*, see section B.8) are grouped into a directory tree which has been packed up into a single binary file for distribution, generally a tar file. The structure of the directory tree is not dictated by **UPS**, but generally it includes (at least) areas for the executables (`bin`), for files accessed by **UPS** tools (`ups`), and for documentation (e.g., `ups/toman`, `docs`, `html`). During product installation on a user node, this file can be unwound using **tar** (optionally with **UPD** as an interface) and installed into its product root directory.

B.4 UPS Databases

A **UPS** database is, simply put, a directory. Within the directory, there is a *product file* for each individual product that is accessible via **UPS** on the system. A product file, described in section B.5, contains identifying information for, and pointers to, each of the installed copies of a product. **UPS** commands refer to the database directory via the environment variable *PRODUCTS*, described in section B.2.

The environment variable *PRODUCTS* can be set to point to one or to several directories, thus allowing support for multiple databases. This allows users to maintain one or more private databases in addition to or instead of the common one(s). Creating additional databases is discussed in GU0014. In **UPS** operations, the databases are searched in the order they appear in *PRODUCTS*.

B.5 UPS Product Files

The information that **UPS** needs in order to identify, locate and retrieve a product and its requirements (requirements are described in section B.11) resides in an ASCII *product file* specific to that product in the **UPS** database. The name of each product file is simply the product's name

(see the list of product files obtained by the command `ls $PRODUCTS` in section B.2). The file identifies the product, the UPS database version, the declared instances of the product and the defined chains. Instance is defined in section B.8, and chain in section B.10.

B.6 Product Versions

UPS supports multiple concurrent versions of software products. Each version of a product is installed and accessed independently of other versions. When a new version of a product becomes available, an existing directory tree is not replaced with another; rather, a branch is added for the new version.

Maintaining multiple, self-contained versions of a product on a single system is often necessary and/or desirable for several reasons. For example, in critical situations like data acquisition, if something goes wrong, you don't want to lose time. **UPS** allows you to back out of a new software version completely and assuredly, and immediately start up a previous tried-and-true version. You can also overlap development and production use of a product, and support many users sharing workstation resources.

B.7 UNIX Operating System Flavors

B.7.1 What is "Flavor"?

Many programs require separate compilation for the different UNIX operating systems. In **UPS**, we maintain different directory trees for the separate compilations (and related files) of the same product. We therefore need to distinguish between OS-dependent compilations. To indicate the OS dependency, we use the term *flavor*. This additional term allows us to maintain the same product name and version across the different operating systems and releases, which is desirable since the same program source files are used in the separate compilations.

On a given system, several different copies of a product may exist. When you use **UPS** commands to manipulate or use a product, the system needs to have enough information to select the appropriate one. Many **UPS** commands support a `-f` option allowing you to specify flavor-related information.

B.7.2 Simple Flavors

Simple flavors (as opposed to extended flavors) are the UNIX operating system names, as returned by the command `funame -s` (for example IRIX or SunOS). This is sufficient for products which can run on multiple releases of a single OS. For products which have no compiled programs, and are thus operating system-independent, a special flavor of NULL is used.

B.7.3 Extended Flavors

In addition to differentiation by OS, some products require separate compilations for different releases of the same operating system (e.g., IRIX+5 and IRIX+6). For this purpose, **UPS** supports *extended flavors*, which are designations that include both OS (simple flavor, e.g., IRIX) and OS release (e.g., +5). This allows a mixed cluster (e.g., IRIX+5 and IRIX+6) to share a single **UPS** database.

Extended flavors may further include information about options used by the product developer at compilation time (e.g., `+debug`).

Extended flavor is designed to accomplish two goals:

- identify the most appropriate compilation of a product for the given operating system and release
- allow the product developer to specify additional compilation options such as **`debug`**, **`optimized`**, etc.; the user can then select and run one of these compiled instances of a product for special purposes

B.8 Instances

Each installed, declared copy of a product in **UPS** is called an *instance* of the product. Each instance has a unique combination of product name, version and (extended) flavor. The concept of *chains*, discussed in section B.10, allows users to easily access the appropriate instance, without needing to remember its version number and (extended) flavor.

B.9 Flavor Specification

When a **UPS** command is issued, the system must determine which product instance or instances to act upon. For some **UPS** commands multiple instances can be retrieved, as is often the case for **`ups list`**, for example. The product name and version are generally specified unambiguously, but the extended flavor can be specified such that it requires some interpretation by the system. This allows **UPS** to do some of the instance selection work for you. In fact for some operations you can choose not to specify flavor at all and let **UPS** determine it completely. If you need to specify the flavor (generally you shouldn't) use the **`-f`** option:

`%` ***ups_command*** **`-f`** ***flavor*** **`[`***other options***`]`** ***product***

The system will then only retrieve an instance exactly matching ***flavor***.

B.10 Chains

We mentioned earlier that **UPS** supports multiple versions of software products on a machine. End users do not find it convenient to specify product version numbers each time they setup a product. This is especially true if product setups are needed at login. Most users want to run the latest, tested, approved version of products without having to keep track of the version numbers.

To allow users to specify the version of a product according to its *status* rather than by its version number, **UPS** supports *chains* to product versions. A chain is a **UPS** database entry (i.e. a clause in a product file) that points to a particular instance declaration in the database (an instance clause in the same product file). It must match the (extended) flavor and version of the instance declaration exactly. It "attaches" a chain name to a product instance, thereby tagging the product instance according to its status.

Five statuses, or chains, have been defined for use: current, new, test, old, and development.

Chain	Option	Usage
current	**`-c`**	default instance recommended for general use
new	**`-n`**	tested instance that is not yet current
test	**`-t`**	instance installed for testing

Chain	Option	Usage
development	`-d`	instance under development
old	`-o`	older instance that was previously current

In **UPS** commands, the command line option associated with the desired chain (status) is used to specify the product instance to retrieve. Using chains is optional, but recommended. Both chained and unchained instances of a product may be declared to **UPS**; the user can still retrieve any instance, chained or not, by using its product version number.

B.11 Product Dependencies (Use and Build Requirements)

Many **UPS** products rely on other products being installed, declared and setup for proper functioning or for use of special features. These are generally referred to as *product dependencies* or *use requirements*[1]. The coupling of products with their dependencies facilitates product setup. You need only setup a single **UPS** product to access any and all of the products listed as its dependencies. Each of the dependencies can be setup in its own right, as well.

Requirements can exist across databases. The databases must be included in *$PRODUCTS* for this to work. When you declare a requirement, **UPS** searches the databases in *$PRODUCTS* and uses the first instance of the required product it finds; you cannot specify the database in which a requirement resides.

Multiple levels of requirements are possible, and use requirements are setup recursively by default. As an example, the mail product **exmh** has several use requirements, one of which is **www**. **www** in turn has use requirements, one of which is **ghostview**. When you setup **exmh**, you also implicitly setup all its use requirements including **www**, and all the next level use requirements including **ghostview**, and so on.

B.12 Notes on Setup and Unsetup

The `setup` command was discussed in section 10.2.2. As simple as it is to use in everyday situations, `setup` contains some subtleties. It is equipped with a host of options that allow you to specify various parameters. We refer you to the man pages for specifics; here we attempt to provide a summary/clarification of the available options. You can specify:

- which chained instance of the product to setup
- which use or build dependencies to setup along with main product (you cannot setup both)
- an option string to pass to the setup script via the environment variable *UPS_OPTIONS*
- enable any optional functionality in the setup script.

By default, the use requirements are setup recursively along with the main product unless the `-j` option is used to restrict the setup to *just* the specified product and none of its dependencies. `setup` also comes equipped with a `-b` option that allows you to setup the build requirements *instead of* the use requirements. Build requirements are not setup recursively; in fact they are not generally distributed with the products. Most users never use `-j` or `-b`.

---

1. There are also such things as *build requirements*. These need to be in place only for building the main product, and are usually omitted on end-user systems.

The last two features are rarely if ever implemented, and require knowledge about the setup script by the user. They are included here for completeness only.

When you unsetup a product, any use-dependent products get deactivated (unsetup) automatically at the same time unless the `-j` or the `-b` option is used:

`-j` unsetup *just* the specified product and none of its dependencies

`-b` unsetup the build requirements of the product (useful only when the product was setup with the `-b` option)

The man pages list other options, but in practice, unsetup scripts aren't coded to exploit them.

Appendix C. Fermi Login Files

This appendix contains file listings of the FUE-customized default login files (the "Fermi Files") used to set up your UNIX environment.[1] If you are on a FUE-compliant system, you are supplied with a copy of each file in your home directory (except for the `fermi.*` and `setup.*` files which are executed directly from `/usr/local/etc`). The files are reprinted here in their entirety except for the copyright disclaimers.

C.1 C Shell Family

C.1.1 .cshrc

The default FUE `.cshrc` file is found in `/usr/local/etc/stdcshrc`.

```
# .cshrc default settings for all users

# @(#) stdcshrc 1.10 Delta: 93/08/19 14:51:27 Extraction 94/07/06 15:16:15 @(#)

# execute fermi.cshrc first
if ( -r /usr/local/etc/fermi.cshrc ) then
        source /usr/local/etc/fermi.cshrc
endif

# Place any items that you want executed even for non-interactive use here

#skip if not interactive shell
if ( $?USER == 0 || $?prompt == 0 ) exit


set noclobber            #prevent overwrite when redirecting output
set ignoreeof            #prevent accidental logouts

# Have mail point to fermimail which is a Berkely mail version. Some packages
# expect mail to be a System V version of mail. If you are having problems
# of this type, you may need to remove the alias.
alias mail          fermimail       #Fermi recommended Mail

#Define various aliases; user selects desired alias by removng the # sign
#alias a            alias
#alias killit       kill -9         #guarantees that a process is killed
#alias h            'history | tail'
#alias ll           ls -l
#alias la           ls -a           #see hidden files
#alias lf           ls -CF          #check file TYPE (exe, dir ..)
#alias rmi          rm -i           #confirm before deletion
#alias home         cd              #HOME
```

1. The `setpath.*` and `setups.*` files reprinted here are part of the FUE product **funkern v5.0**. All other files are part of FUE's **systools v4.3**.

```
#alias side         'cd ../\!*'     #side
#alias down         'cd \!*'        #down
#alias up           cd ..           #up
#alias cpi          cp -i           #no overwrite of output file
#alias cd          'cd \!*;echo $cwd'
#alias mvi          mv -i           #confirm before moving
#Next alias replaces standard info command on SGI platforms
#alias info         Info            #get list of info articles
#
#VMS type commands
#
#alias dir          ls -l
#alias copy         cp
#alias rename       mv
```

`.cshrc` executes `/usr/local/etc/fermi.cshrc`.

C.1.2 fermi.cshrc

The default FUE `fermi.cshrc` file is found in `/usr/local/etc/fermi.cshrc`.

```
# @(#)' fermi.cshrc 4.32 Delta: 94/09/29 22:22:46 Extraction 94/09/29 22:34:39 @(
#)

# fermi.cshrc settings for all users; will be called by the default .cshrc

set MACH_ID=`/usr/local/bin/funame -n`
set MACH_TYPE=`/usr/local/bin/funame -m`
set MACH_OS=`/usr/local/bin/funame -s`

#Determine if this is the first time the fermi.cshrc file has been executed
#on this machine
if ($?FERMICSHRC) then
        if ( $FERMICSHRC == "$MACH_ID" ) then
                set firsttime=0
        else
                set firsttime=1
                setenv FERMICSHRC "$MACH_ID"
        endif
else
        setenv FERMICSHRC "$MACH_ID"
        set firsttime=1
endif

#Execute the things that are only done the first time through
if ( $firsttime == "1" ) then
        # set path
        if ( -r /usr/local/etc/setpath.csh ) then
                source /usr/local/etc/setpath.csh
        endif
        #
        # Establish MANPATH
        #
        setenv MANPATH /usr/products/catman:/usr/products/man:/afs/fnal/products
/catman:/afs/fnal/products/man:/usr/catman:/usr/man:/usr/share/catman:/usr/share
/man:/usr/local/catman:/usr/local/man

endif

#Execute the items done each time (basically only aliases)

#
# Setup UPS
```

```
#
if ( -r /usr/local/etc/setups.csh ) then
        source /usr/local/etc/setups.csh
endif

switch ($MACH_OS)
case IRIX:
        alias fermimail '/usr/sbin/Mail'        # Berkely mail
breaksw
case SunOS:
        alias fermimail '/usr/ucb/Mail'         # Berkely mail
        alias man       'man -F'
breaksw
case AIX:
        alias fermimail '/usr/ucb/Mail'                 #Berkely Mail
breaksw
case ULTRIX:
        alias fermimail '/usr/ucb/Mail'                 # Berkely Mail
breaksw
case OSF1:
        alias fermimail '/usr/bin/Mail'                 # Berkely Mail
breaksw
case HP-UX:
        alias fermimail '/usr/bin/mailx'        #Berkely Mail
breaksw
default
        if ( -x /usr/ucb/Mail ) then
                alias fermimail '/usr/ucb/Mail'         #Berkely Mail
                endif
        if ( -x /usr/sbin/Mail ) then
                alias fermimail '/usr/sbin/Mail'        #Berkely Mail
        endif
        if ( -x /usr/bsd/Mail ) then
                alias fermimail '/usr/bsd/Mail'         #Berkely Mail
        endif
        if ( -x /usr/bin/mailx ) then
                alias fermimail '/usr/bin/mailx'        #Berkely Mail
        endif
        if ( -x /usr/bin/Mail ) then
                alias fermimail '/usr/bin/Mail'        #Berkely Mail
        endif
breaksw
endsw
#
# Check for existance of local.cshsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if ( -r /usr/local/etc/local.cshrc ) then
        source /usr/local/etc/local.cshrc
endif
```

`fermi.cshrc` calls `/usr/local/etc/setpath.csh` to set the shell variable *path*, and it executes `/usr/local/etc/setups.csh` to set up **ups**.

C.1.3 setpath.csh

The default FUE `setpath.csh` file is found in `/usr/local/etc/setpath.csh`.

```
# @(#) setpath.csh 1.7 Delta: 95/03/22 22:13:29 Extraction 95/03/22 22:13:53 @(#
)

if ($?HOME == 0 ) then
        setenv HOME /
```

```
endif

set path=""
foreach DIR ( /usr/sbin \
        /opt/SUNWspro/bin \
        /usr/ccs/bin \
        /usr/lang \
        /usr/bsd \
        /bin \
        /usr/bin \
        /usr/lbin \
        /usr/ucb \
        /etc \
        /usr/etc \
        /usr/afsws/bin \
        /usr/openwin/bin \
        /usr/bin/X11 \
        /usr/kinet/bin \
        /usr/local/bin \
        /usr/bin/mh \
        /usr/sccs )
        if ( -d $DIR ) then
                set path=($path $DIR)
        endif
end
#
# Add your HOME/bin and bin."flavor"
# Put your Sun executables in a bin named bin.SunOS,
#     your IRIX executables in bin.IRIX,
#     your AIX     "         " bin.AIX,
#  ...etc...
#
# Do not do this if you are root, because //bin is your
# $HOME/bin, and do not put current directin in root's path
if ($?MACH_OS == 0) then
        set MACH_OS=`funame -s`
endif
if ( "$HOME" != "/" ) then
        foreach DIR ( $HOME/bin.$MACH_OS $HOME/bin )
                if ( -d $DIR ) then
                        set path=($DIR $path)
                endif
        end
        set path=($path .)
endif
```

C.1.4 setups.csh

The default FUE `setups.csh` file is found in `/usr/local/etc/setups.csh`.

```
#Set environment for UPS
if (! -r ~/.noupsproducts) then #start if .noupsproducts
        set MACH_OS=`funame -s`
        set setprod=no
        if ( $?PRODUCTS == 0 ) then
                set setprod=yes
        else
                 set fchar=`echo $PRODUCTS | cut -c1`
                 if ( "x/" != x"$fchar" ) then
                        set setprod=yes
                 endif
        endif
        if ( $setprod == "yes" ) then #start if $PRODUCTS already set
           if ( $?UPS_EXTRA_DIR == 0 ) then
```

```
            set UPS_EXTRA_DIR=""
        endif
            unsetenv PRODUCTS
            # Add other directories areas that might be product databases
            # to the foreach list
            foreach PROD ( $UPS_EXTRA_DIR \
                    /`funame -n`/products/ups_database/declared \
                    `logdir products`/ups_database/declared \
                    /usr/products/ups_database/declared \
                    /afs/fnal/products/ups_database/declared)
                    if (-d $PROD) then
                            if ($?PRODUCTS) then
                                    set exists=`echo $PRODUCTS | grep -c $PR
OD`
                                    if ( $exists == "0" ) then
                                            setenv PRODUCTS "$PRODUCTS $PROD
"
                                    endif
                            else
                                    setenv PRODUCTS $PROD
                            endif
                    endif
            end
    endif #end if $PRODUCTS already set

    if ($?PRODUCTS) then #set if $PRODUCTS set
            set setups=no
            if ( $?UPS_DIR == 0 ) then
                    set setups=yes
            else
                    set fchar=`echo $UPS_DIR | cut -c1`
                    if ( "x/" != x"$fchar" ) then
                            set setups=yes
                    endif
            endif
            if ( $setups == "yes" ) then #start if $UPS_DIR already set
                    set setups=no
                    set full=${MACH_OS}+`funame -r`
                    foreach PROD ( $PRODUCTS )
                    if ( -r $PROD/ups ) then
                            if ( `grep -c "current.$MACH_OS" $PROD/ups` == 0
) then
                                    continue
                            endif
                            foreach i ( `grep "current.$MACH_OS" $PROD/ups|a
wk '{print $2}'|sort -r` )
                                    if ( $full =~ $i* ) then
                                            set flavor=$i
                                            break
                                    endif
                            end
                            if ( $?flavor == 0 ) then
                                    continue
                            endif
#characater after ${flavor} and before " is tab (next 2 lines)
                            set CURRENT=`grep "current.${flavor}    " $PROD/
ups|awk '{print $3}'`
                            setenv UPS_DIR `grep "^Instance:.*${flavor}
" ${PROD}/ups|grep \"${CURRENT}\"|awk '{print $4}'`
                            set fchar=`echo $UPS_DIR | cut -c1`
                            if ( "x/" == x"$fchar" ) then
                                    set setups=yes
                                    break
                            endif
                    endif
```

```
                        end
                        if ($?UPS_DIR) then
                                if ( -r $UPS_DIR/ups/ups_init ) then
                                        source $UPS_DIR/ups/ups_init
                                        setup ups
                                endif
                        endif
                endif #end if $UPS_DIR already set
#Even though UPS_DIR may have already been setup, aliases aren't always
# passed along, thus we need to always do the source of ups_init and
# the setup.csh file. We can't simply do a setup of ups, because then you
# get the current version of ups instead of the version of UPS when you
# entered this process. The setup of ups relies on UPS_PROD_VERSION being
# set (although not in a serious way). This process assumes the version is
# the last component of the $UPS_DIR directory. This isn't guaranteed to be
# correct, but will work ok for this situation.
                if ($?UPS_DIR) then #start if $UPS_DIR is set
                        if ( -r $UPS_DIR/ups/ups_init ) then
                                setenv UPS_PROD_VERSION `basename $UPS_DIR`
                                source $UPS_DIR/ups/ups_init
                                if ( -r $UPS_DIR/ups/setup.csh ) then
                                        source $UPS_DIR/ups/setup.csh
                                endif
                                unsetenv UPS_PROD_VERSION
                        else
                                echo '$UPS_DIR set, but $UPS_DIR/ups/ups_init do
esnt exist, ups environment not setup'
                        endif
                else
                        echo 'Unable to set $UPS_DIR, ups environment not setup'
                endif #end if $UPS_DIR set
        else
                echo 'Unable to set $PRODUCTS, ups environment not setup'
        endif #end if $PRODUCTS set

endif #end if .noupsproducts
```

C.1.5 .login

The default FUE `.login` file is found in `/usr/local/etc/stdlogin`.

```
# .login default settings for all users

#@(#) stdlogin 1.6 Delta: 92/11/05 14:17:45 Extraction 94/07/06 15:16:16 @(#)

#
# .login default settings for all users
#
# execute fermi.login first
#
if ( -r /usr/local/etc/fermi.login ) then
        source /usr/local/etc/fermi.login
endif
#
# defines alias for cedit, comment out if not wanted
#
if ( ! -e ~/.noupsproducts ) then
        setup cedit
endif
#
# Next line sets prompt to <machine_name>
#
set prompt="<$MACH_ID> "
```

```
#
# The standard fermi.profile does not attempt to include /usr/5bin
# (in SunOS) in your PATH.  /usr/5bin contains the UNIX System V version
# of several commands.  By including /usr/5bin first, you will get the
# System V version before the BSD version.  Uncomment one of the following
# if you so desire:
#
#set path=(/usr/5bin $path)
#set path=($path /usr/5bin)
#
# The savehist & history settings maybe expanded to 100
#
set savehist=20       #save last 20 commands for next session
set history=20        #retain the last 20 commands
#
# Uncomment the following to change...
#
#stty kill '^u'          # sets line kill to <ctrl-u>
#stty erase '^H'  # sets erase to backspace
#stty intr '^?'   # sets interrupt to delete
```

The `.login` file executes the file `/usr/local/etc/fermi.login`.

C.1.6 fermi.login

The default FUE `fermi.login` file is found in `/usr/local/etc/fermi.login`.

```
# @(#) fermi.login 4.31 Delta: 94/09/29 22:31:20 Extraction 94/09/29 22:34:39 @(
#)

#
# fermi.login settings for all users; called by the default .login
# Set the umask so that newly created files and directories will be readable
# by others, but writable only by the user.
#
umask 022
#
# Following put in to handle NQS
#
if ( $?ENVIRONMENT ) then
        if ( $ENVIRONMENT == "BATCH" ) exit
endif
#
# Determine terminal type
#
set ttype=`echo $term |cut -c1`
switch (x$term)
case x:
case xunknown:
case xarpanet:
case xnetwork:
case xnet:
case xdialup:
case xdumb:
        set term=vt100
        stty erase '^?'
breaksw
default
        if ( -r /usr/lib/terminfo/$ttype/$term || -r /usr/share/lib/terminfo/$tt
ype/$term ) then
                switch ($term)
                case iris-ansi:
                case iris-ansi-net:
```

```
                case hp:
                        stty erase '^h'
                breaksw
                default
                        stty erase '^?'
                breaksw
                endsw
        else
                set bterm=`echo $term | cut -c1-3`
                if ( -r /usr/lib/terminfo/$ttype/$bterm || -r /usr/share/lib/ter
minfo/$ttype/$bterm ) then
                        set term=$bterm
                        stty erase '^?'
                else
                        switch ($MACH_OS)
                        case IRIX:
                        case SunOS:
                        case AIX:
                                switch ($bterm)
                                case vt2:
                                case vt3:
                                        set term=vt220
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        case ULTRIX:
                                switch ($bterm)
                                case vt3:
                                        set term=vt300
                                        stty erase '^?'
                                breaksw
                                case vt2:
                                        set term=vt200
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        case OSF1:
                                switch ($bterm)
                                case vt2:
                                case vt3:
                                        stty erase '^?'
                                breaksw
                                default
                                        set term=vt100
                                        stty erase '^?'
                                breaksw
                                endsw
                        breaksw
                        default
                                set term=vt100
                                stty erase '^?'
                        breaksw
                        endsw
                endif
        endif
```

```
breaksw
endsw
#
# Set DISPLAY
#
if ( ! $?DISPLAY ) then
        set TTYPORT=`tty`
        if ( $TTYPORT == /dev/console ) then
                setenv DISPLAY "localhost:0"
        else if ( $?REMOTEHOST ) then
                setenv DISPLAY "${REMOTEHOST}:0"
        else
                set TTYNAME=`echo $TTYPORT |cut -c6-`
                set REMOTEHOST=`who|grep "$TTYNAME"|awk '{print $6}'|sed 's/(//'
|sed 's/)//'`
                setenv DISPLAY "${REMOTEHOST}:0"
        endif
endif
#
# Who are you?
#
if ( $?LOGNAME ) then
        setenv LOGNAME "$LOGNAME"
else if ( $?USER ) then
        setenv LOGNAME "$USER"
else if ( $?REMOTEUSER ) then
        setenv LOGNAME "$REMOTEUSER"
endif
#
# The switch statement is to set the mail variable
# based on machine type.  Also motd and news are
# dependent on system type. Some systems automatically
# display motd (and thus shouldn't be repeated here.
# Others don't have news.
#
switch ($MACH_OS)
case IRIX:
        setenv TZ CST6CDT
        if ( `funame -r|cut -c1` == 3 ) then
                set mail=/usr/mail/$LOGNAME
                if ( { /bin/mail -e } ) then
                        echo 'You have mail.'
                endif
        endif
        if ( { test -d /usr/news } ) then
                /usr/bin/news
        endif
breaksw
case SunOS:
        setenv TZ CST6CDT
        set mail=/usr/spool/mail/$LOGNAME
breaksw
case AIX:
        setenv TZ CST6CDT
        /bin/news
        setenv MAIL "/usr/spool/mail/$LOGNAME"
        setenv MAILMSG "[YOU HAVE NEW MAIL]"
        if ( { test -s $MAIL } ) then
                echo 'You have mail.'
        endif
        setenv EMULATE none
breaksw
case ULTRIX:
        set mail=/usr/spool/mail/$USER
breaksw
```

```
case OSF1:
        /usr/bin/news
        set mail=/usr/spool/mail/$USER
breaksw
case HP-UX:
        setenv TZ CST6CDT
        cat -s /etc/motd
        /usr/bin/news
        set mail=/usr/mail/$LOGNAME
        if ( { /bin/mail -e } ) then
                echo 'You have mail.'
        endif
breaksw
case "":
        #If for some reason funame -s returns NULL, as opposed to returning
        #something that isn't recognized, do nothing
breaksw
default
        cat -s /etc/motd
        if ( -x /usr/bin/news) then
                /usr/bin/news
        endif
        if ( -x /bin/news) then
                /bin/news
        endif
        if ( -d /usr/spool/mail ) then
                set mail=/usr/spool/mail
        endif
        if (    -d /usr/mail ) then
                set mail=/usr/mail/$LOGNAME
        endif
        if ( -x /bin/mail ) then
                if ( { /bin/mail -e } ) then
                        echo 'You have mail.'
                endif
        endif
breaksw
endsw
#
# Establish PAGER
#
if ( -r /usr/local/bin/less ) then
        setenv PAGER /usr/local/bin/less
else
        setenv PAGER more
endif
#
# Common terminal characteristics
#
stty intr '^c'  # set interrupt key to <ctrl-c>
stty kill '^x'  # set kill key to <ctrl-x>
stty echoe                      # erase ERASEd characters
echo "Terminal Type is $TERM"
/usr/local/bin/Info -new
#
# Some systems the user doesn't own his tty device ( Sun OpenWindows) so
# redirect stderr
#
/bin/chmod 622 `tty` >& /dev/null
#
# Check for existance of local.cshsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if ( -r /usr/local/etc/local.login ) then
```

```
        source /usr/local/etc/local.login
endif
```

C.2 Bourne Shell Family

C.2.1 .profile

The default FUE `.profile` file is found in `/usr/local/etc/stdprofile`.

```
# @(#) stdprofile 1.6 Delta: 92/11/05 14:19:26 Extraction 94/07/06 15:16:17 @(#)

#
# .profile default settings for all users
#
# Execute fermi.profile first!
#
if [ -r /usr/local/etc/fermi.profile ]
then
  . /usr/local/etc/fermi.profile
fi
#
# PS1 sets prompt to <machine_name>
# PS2 sets secondary prompt to <more>
#
export PS1;             PS1="<${MACH_ID}> "
export PS2;             PS2="<more> "
#
# HISTSIZE is the number of commands retained in history
#
export HISTSIZE;        HISTSIZE=20       #retain the last 20 commands
#
# EDITOR is used as your default editor, emacs or vi typically
# VISUAL is used as your command line editor, emacs or vi typically
# To use the emacs version, comment out or remove the vi references
# and uncomment the emacs verisons...
#
#export EDITOR;         EDITOR=emacs
#export VISUAL;         VISUAL=emacs
#
export EDITOR;          EDITOR=vi
export VISUAL;          VISUAL=vi
#
# The standard fermi.profile does not attempt to include /usr/5bin
# (in SunOS) in your PATH.  /usr/5bin contains the UNIX System V version
# of several commands.  By including /usr/5bin first, you will get the
# System V version before the BSD version.  Uncomment one of the following
# if you so desire:
#
# PATH=/usr/5bin:$PATH
# PATH=$PATH:/usr/5bin
#
# ENV is the location of your shell alias file
#
export ENV;             ENV=~/.shrc
#
# Uncomment the following to change...
#
#stty kill ^U   # sets line kill to <ctrl>U
#stty erase  # sets erase to backspace
#stty intr ^?   # sets interrupt to delete
#
```

```
if [ "`basename $SHELL|sed 's/[0-9]//'`" = "sh" ] && [ -r $HOME/.shrc ]
then
  . $HOME/.shrc
fi
```

This file calls `/usr/local/bin/fermi.profile`.

C.2.2 fermi.profile

The default FUE `fermi.profile` file is found in `/usr/local/etc/fermi.profile`.

```
# @(#) fermi.profile 1.33 Delta: 94/10/31 22:14:25 Extraction 94/11/02 16:56:08
@(#)

# fermi.profile settings for all users; called by the user bourne/korn .profile
# Set the umask so that newly created files and directories will be readable
# by others, but writable only by the user.
umask 022
MACH_ID="`/usr/local/bin/funame -n`"
MACH_TYPE="`/usr/local/bin/funame -m`"
MACH_OS="`/usr/local/bin/funame -s`"
#
# Set the PATH appropriately
#
if [ -r /usr/local/etc/setpath.sh ]
then
  . /usr/local/etc/setpath.sh
fi

#
# Set a respectable MANPATH
#
MANPATH=/usr/products/catman:/usr/products/man:/afs/fnal/products/catman:/afs/fn
al/products/man:/usr/catman:/usr/man:/usr/share/catman:/usr/share/man:/usr/local
/catman:/usr/local/man
#
# Following put in to handle NQS
# Only do rest if non-interactive session
#
if [ "$ENVIRONMENT" != "BATCH" ] && [ "x`logname`" != "x" ]
then
#
# Determine the terminal type
#
TTYPE=`echo $TERM | cut -c1`
case $TERM in
""|arpanet|net|network|dialup|unknown|dumb)
        TERM=vt100
        stty erase '^?'
;;
*)
        if [ -r /usr/lib/terminfo/$TTYPE/$TERM -o -r /usr/share/lib/terminfo/$TT
YPE/$TERM ]
        then
                case $TERM in
                iris-ansi*|hp)
                        stty erase '^h'
                ;;
                *)
                        stty erase '^?'
                ;;
                esac
        else
```

```
                BTERM=`echo $TERM | cut -c1-3`
                if [ -r /usr/lib/terminfo/$TTYPE/$BTERM -o -r /usr/share/lib/ter
minfo/$TTYPE/$BTERM ]
                then
                        TERM=$BTERM
                        stty erase '^?'
                else
                        case $MACH_OS in
                        IRIX|SunOS|AIX)
                                case $BTERM in
                                vt[2|3])
                                        TERM=vt220
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        ULTRIX)
                                case $BTERM in
                                vt3)
                                        TERM=vt300
                                        stty erase '^?'
                                ;;
                                vt2)
                                        TERM=vt200
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        OSF1)
                                case $BTERM in
                                vt[2|3])
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac
                        ;;
                        HP-UX)
                                case $BTERM in
                                vt3)
                                        TERM=vt320
                                        stty erase '^?'
                                ;;
                                vt2)
                                        TERM=vt200
                                        stty erase '^?'
                                ;;
                                *)
                                        TERM=vt100
                                        stty erase '^?'
                                ;;
                                esac

                        ;;
                        *)
```

```
                                TERM=vt100
                                stty erase '^?'
                        ;;
                        esac
                fi
        fi
;;
esac
#
# Set the DISPLAY
#
if [ x"$DISPLAY" = x ]
then
        TTYPORT=`tty`
        if [ "$TTYPORT" = "/dev/console" ]
        then
                DISPLAY="localhost:0"
        elif [ "$REMOTEHOST" = "" ]
        then
                TTYNAME=`echo $TTYPORT | cut -c6-`
                REMOTEHOST=`who|grep "$TTYNAME"|awk '{print $6}'|sed 's/(//'|sed
 's/)//'`
                DISPLAY="${REMOTEHOST}:0"
        else
                DISPLAY="${REMOTEHOST}:0"
        fi
fi
#
# Who are you?
#
if [ "x$LOGNAME" = "x" ]
then
        if [ "x$USER" = "x" ]
        then
        if [ "x$REMOTEUSER" != "x" ]
        then
                LOGNAME="$REMOTEUSER"
        fi
        else
                LOGNAME="$USER"
        fi
fi
#
# The case statement is to set the mail variable
# based on machine type.  Also motd and news are
# dependent on system type. Some systems automatically
# display motd (and thus shouldn't be repeated here.
# Others don't have news.
#
case $MACH_OS in
IRIX)
        TZ=CST6CDT
        if [ "`funame -r | cut -c1`" = "3" ]
        then
                MAIL=/usr/mail/$LOGNAME
                /bin/mail -e
                if [ $? = 0 ]
                then
                        echo 'You have mail.'
                fi
        fi
        if [ -d /usr/news ]
        then
                /usr/bin/news
        fi
```

```
;;
SunOS)
        TZ=CST6CDT
        MAIL=/usr/spool/mail/$LOGNAME
;;
AIX)
        TZ=CST6CDT
        /bin/news
        MAIL="/usr/spool/mail/$LOGNAME"
        export MAILMESG;MAILMESG="[YOU HAVE NEW MAIL]"
        if [ -s $MAIL ]
        then
                echo 'You have mail.'
        fi
        EMULATE=none
        export EMULATE
;;
ULTRIX)
        MAIL=/usr/spool/mail/$USER
;;
OSF1)
        /usr/bin/news
        MAIL=/usr/spool/mail/$USER
;;
HP_UX)
        TZ=CST6CDT
        cat -s /etc/motd
        MAIL=/usr/mail/$LOGNAME
        /bin/mail -e
        if [ $? = 0 ]
        then
                echo 'You have mail.'
        fi
        /usr/bin/news
;;
"")
        # Catch for the "funame returns null" bug
        :
;;
*)
        cat -s /etc/motd
        if [ -x /usr/bin/news ]
        then
                /usr/bin/news
        fi
        if [ -x /bin/news ]
        then
                /bin/news
        fi
        if [ -d /usr/spool/mail ]
        then
                MAIL=/usr/spool/mail
        fi
        if [    -d /usr/mail    ]
        then
                MAIL=/usr/mail/$LOGNAME
        fi
        if [ -x /bin/mail ]
        then
                /bin/mail -e
                if [ $? = 0 ]
                then
                        echo 'You have mail.'
                fi
        fi
```

```
;;
esac
#
#  Establish PAGER
#
if [ -x /usr/local/bin/less ]
then
        PAGER=/usr/local/bin/less
else
        PAGER=more
fi
#
# Common terminal characteristics
#
stty intr '^c'  # set interrupt key to <ctrl-c>
stty kill '^x'  # set kill key to <ctrl-x>
stty echoe                      # erase ERASEd characters
echo "Terminal Type is $TERM"
if [ -x /usr/local/bin/Info ]
then
        /usr/local/bin/Info -new
fi
# Some systems the user doesn't own his tty device ( Sun OpenWindows) so
# redirect stderr
/bin/chmod 622 `tty` >/dev/null 2>&1
export DISPLAY MAIL TZ TERM PATH PAGER MANPATH

fi      # This is end of if statement for interactive use.
#
# Check for existance of local.profile file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if [ -r /usr/local/etc/local.profile ]
then
  . /usr/local/etc/local.profile
fi
```

`fermi.profile` executes `/usr/local/etc/setpath.sh` to set the *path* variable.

C.2.3 setpath.sh

The default FUE `setpath.sh` file is found in `/usr/local/etc/setpath.sh`.

```
# @(#) setpath.sh 1.9 Delta: 95/01/18 15:29:38 Extraction 95/03/22 22:13:53 @(#)

if [ "x"$HOME = "x" ]
then
        HOME=/
fi

PATH=""
for DIR in /usr/sbin \
                                        /opt/SUNWspro/bin \
                                        /usr/ccs/bin \
                                        /usr/lang \
                                        /usr/bsd \
                                        /bin \
                                        /usr/bin \
                                        /usr/lbin \
                                        /usr/ucb \
                                        /etc \
                                        /usr/etc \
                                        /usr/afsws/bin \
```

```
                                        /usr/openwin/bin \
                                        /usr/bin/X11 \
                                        /usr/kinet/bin \
                                        /usr/local/bin \
                                        /usr/bin/mh \
                                        /usr/sccs
do
        if [ -d ${DIR} ]
        then
                PATH=${PATH}${DIR}:
        fi
done
#
# zsh bug fix
#
PATH=`echo $PATH|/bin/sed 's/\.//g'`
if [ "x"$MACH_OS = "x" ]
then
        MACH_OS=`funame -s`
fi
# do not do this if you are root, because //bin is your $HOME/bin, and
# do not put current directory in root's path
if [ "$HOME" != "/" ]
then
        #
        # Add your HOME/bin and bin."flavor"
        # Put your Sun executables in a bin named bin.SunOS,
        #     your IRIX executables in bin.IRIX,
        #     your AIX      "          " bin.AIX,
        #  ...etc...
        #
        for DIR in ${HOME}/bin.${MACH_OS} $HOME/bin
        do
                if [ -d ${DIR} ]
                then
                        PATH=${DIR}:${PATH}
                fi
        done
        PATH=${PATH}.:
fi

export PATH
```

C.2.4 .shrc

The default FUE `.shrc` file is found in `/usr/local/etc/stdshrc`.

```
# .shrc default settings for all users

#execute fermi.shrc first
if [ -r /usr/local/etc/fermi.shrc ]
then
        . /usr/local/etc/fermi.shrc
fi

#Place items that you want run even when not interactive shell here

#skip if not interactive shell
if [ "x$USER" = "x" ] || [ "x$PS1" = "x" ]
then
        :
else
```

```
        if [ "`basename $SHELL|sed 's/[0-9]//'`" != "sh" ]
        then
          set -o noclobber            #prevent overwrite when redirecting output
          set -o ignoreeof            #prevent accidental logouts

  # Have mail point to fermimail which is a Berkely mail version. Some packages
  # expect mail to be a System V version of mail. If you are having problems
  # of this type, you may need to remove the alias.
        alias mail='fermimail'

  #Define various aliases; user selects desired alias by removng the # sign
  # ONLY IF YOUR SHELL SUPPORTS ALIASING!
          #alias        a='alias'
          #alias        logout='exit'
          #alias        killit='kill -9'        # guarantees that a process is k
illed
          #alias        ll='ls -l'              # generate a long listing
          #alias        la='ls -a'              # see hidden files
          #alias        lf='ls -CF'             # check file TYPE (exe, dir ..)
          #alias        rmi='rm -i'             # confirm before deletion
          #alias        cpi='cp -i'             # no overwrite of output file
          #alias        mvi='mv -i'             # confirm before moving
          #alias        more='less'             # when you explicitly "more" a f
ile

  #Next alias replaces standard info command on SGI platforms
          #alias        info='Info'            #get list of info articles

        #VMS type commands
          #alias        dir='ls -l'
          #alias        copy='cp'
          #alias        rename='mv'

        else
  # Have mail point to fermimail which is a Berkely mail version. Some packages
  # expect mail to be a System V version of mail. If you are having problems
  # of this type, you may need to remove the alias.
          mail ()
          {
            fermimail "$@"
          }
  #Define various functions; user selects desired functions by removng the # sig
n
  #Provided for shells that do not support aliasing.
          #killit ()
          #{
            #  kill -9  # guarantees that a process is killed
          #}
          #ll ()
          #{
            #  ls -l    "$@"     # generate a long listing
          #}
          #la ()
          #{
            #  ls -a    "$@"     # see hidden files
          #}
          #lf ()
          #{
            #  ls -CF "$@"      # display file type, columnwise
          #}
          #rmi ()
          #{
            #  rm -i "$@"       # confirm before delete
          #}
```

```
        #cpi ()
        #{
          #  cp -i "$@"        # no overwrite of output file (see also set nocl
obber)
        #}
        #mvi ()
        #{
          #  mv -i "$@"        # confirm before moving
        #}
        #more ()
        #{
          #  less "$@"     # when you explicitly "more" a file
        #}

  #Next function replaces standard info command on SGI platforms
        #info ()
        #{
          #  Info "$@"         # get list of info articles
        #}

  #VMS type commands
        #dir ()
        #{
          #  ls -l "$@"
        #}
        #copy ()
        #{
          #  cp "$@"
        #}
        #rename ()
        #{
          #  mv "$@"
        #}
      fi
fi
```

The `.shrc` file calls `/usr/local/etc/fermi.shrc`.

C.2.5 fermi.shrc

The default FUE `fermi.shrc` file is found in `/usr/local/etc/fermi.shrc`.

```
# @(#) fermi.shrc 1.10 Delta: 94/09/29 22:22:52 Extraction 94/09/29 22:34:43 @(#
)

# fermi.shrc settings for all users; will be called by the default .profile

#
# Setup UPS
#
if [ -r /usr/local/etc/setups.sh ]
then
        . /usr/local/etc/setups.sh
fi

#Do machine dependent functions:
case $MACH_OS in
IRIX)
  fermimail ()
  {
    /usr/sbin/Mail "$@"
  }
;;
SunOS)
```

```
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
  man ()
  {
        /usr/bin/man -F "$@"
  }
;;
AIX)
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
;;
ULTRIX)
  fermimail ()
  {
    /usr/ucb/Mail "$@"
  }
;;
OSF1)
  fermimail ()
  {
    /usr/bin/Mail "$@"
  }
;;
HP-UX)
  fermimail ()
  {
    /usr/bin/mailx "$@"
  }
;;
*)
  if [ -x /usr/ucb/Mail ]
  then
    fermimail ()
    {
      /usr/ucb/Mail "$@"
    }
  elif [ -x /usr/sbin/Mail ]
  then
    fermimail ()
    {
      /usr/sbin/Mail "$@"
    }
  elif [ -x /usr/bsd/Mail ]
  then
    fermimail ()
    {
      /usr/bsd/Mail "$@"
    }
  elif [ -x /usr/bin/mailx ]
  then
    fermimail ()
    {
      /usr/bin/mailx "$@"
    }
  elif [ -x /usr/bin/Mail ]
  then
    fermimail ()
    {
      /usr/bin/Mail "$@"
    }
  fi
```

```
;;
esac
#
# Check for existance of local.shsrc file. This is the file admins should
# put in any changes, additions, etc. so that they don't need to re-edit this
# file every release.
#
if [ -r /usr/local/etc/local.shrc ]
then
  . /usr/local/etc/local.shrc
fi
```

`fermi.shrc` executes `/usr/local/etc/setups.sh`.

C.2.6 setups.sh

The default FUE `setups.sh` file is found in `/usr/local/etc/setups.sh`.

```
# @(#) setups.sh 2.2 Delta: 95/01/09 15:48:34 Extraction 95/03/22 22:13:54 @(#)

#Set environment for UPS
if [ ! -r $HOME/.noupsproducts ] #start if .noupsproducts
then
        MACH_OS=`funame -s`
        fchar=`echo ${PRODUCTS:-}|cut -c1`
        if [ "x$fchar" != "x/" ] #start check if $PRODUCTS already set
        then
                prodset=no
                unset PRODUCTS
                # Add any other directories that are UPS databases in the
                # for list
                for PROD in $UPS_EXTRA_DIR \
                    /`funame -n`/products/ups_database/declared \
                        `logdir products`/ups_database/declared \
                        /usr/products/ups_database/declared \
                        /afs/fnal/products/ups_database/declared
                do
                        if [ -d $PROD ]
                        then
                                if [ "$prodset" = "no" ]
                                then
                                        PRODUCTS=$PROD
                                else
                                        exists=`echo $PRODUCTS | grep -c $PROD`
                                        if [ $exists = "0" ]
                                        then
                                                PRODUCTS="$PRODUCTS $PROD"
                                        fi
                                fi
                                prodset=yes
                        fi
                done
                export PRODUCTS
        else
                prodset=yes
        fi #end check if $PRODUCTS already set
        if [ "$prodset" = "yes" ] #start if $PRODUCTS is set
        then
                fchar=`echo ${UPS_DIR:-}|cut -c1`
                if [ "x$fchar" != "x/" ] #start if $UPS_DIR already set
                then
                        setups=no
                        full=${MACH_OS}+`funame -r`
```

```
                        for PROD in $PRODUCTS
                        do
                        if [ -f $PROD/ups ]
                        then
                                if [ `grep -c "current.$MACH_OS" $PROD/ups` -eq
0 ]
                                then
                                        continue
                                fi
                                flavor=x
                                for i in `grep "current.$MACH_OS" $PROD/ups|awk
'{print $2}'|sort -r`
                                do
                                        case "$full" in
                                        $i* )   flavor=$i
                                                break;;
                                        *) ;;
                                        esac
                                done
                                if [ $flavor = x ]
                                then
                                        continue
                                fi
#character after ${flavor} and before quote is tab (next two lines)
                                CURRENT=`grep "current.${flavor}        " $PROD/
ups|awk '{print $3}'`
                                UPS_DIR=`grep "^Instance:.*${flavor}    " ${PROD
}/ups |grep \"${CURRENT}\"|awk '{print $4}'`
                        fi
                        fchar=`echo ${UPS_DIR:-}|cut -c1`
                        if [ "x$fchar" = "x/" ]
                        then
                                setups=yes
                                break
                        fi
                        done
                        if [ "$setups" = "yes" ]
                        then
                                export UPS_DIR
                                if [ -f $UPS_DIR/ups/ups_init.sh ]
                                then
                                        . $UPS_DIR/ups/ups_init.sh
                                        setup ups
                                fi
                        fi
                else
                        setups=yes
                fi #end if $UPS_DIR already set
#Even though UPS_DIR may have already been setup, aliases aren't always
# passed along, thus we need to always do the source of ups_init and
# the setup.sh file. We can't simply do a setup of ups, because then you
# get the current version of ups instead of the version of UPS when you
# entered this process. The setup of ups relies on UPS_PROD_VERSION being
# set (although not in a serious way). This process assumes the version is
# the last component of the $UPS_DIR directory. This isn't guaranteed to be
# correct, but will work ok for this situation.
                if [ "$setups" = "yes" ]
                then
                        if [ -f $UPS_DIR/ups/ups_init.sh ]
                        then
                                . $UPS_DIR/ups/ups_init.sh
                                UPS_PROD_VERSION=`basename $UPS_DIR`
                                if [ -f $UPS_DIR/ups/setup.sh ]
                                then
                                        . $UPS_DIR/ups/setup.sh
```

```
                                fi
                                unset UPS_PROD_VERSION
                        else
                                echo "\$UPS_DIR set, but \$UPS_DIR/ups/ups_init.
sh doesn't exist. No ups environment setup."
                        fi
                else
                        echo "Unable to set \$UPS_DIR, ups environment not setup
"
                fi
        else
                        echo "Unable to set \$PRODUCTS, ups environment not setup"
        fi #end if $PRODUCTS is set
fi #end if .noupsproducts
```

Appendix D. awk's Programming Model

This appendix is adapted from a section of the same name in the book *sed & awk*, published by O'Reilly & Associates. It describes the generic structure and organization of an **awk** program.

An **awk** program consists of what is called a *main input loop*. A loop is a routine that is executed over and over again until some condition exists that terminates it. You don't write this loop, it is given; it exists as the framework within which the code that you do write will be executed. The main input loop in **awk** is a routine that reads one line of input from a file and makes it available for processing. The actions you write to do the processing assume that there is a line of input available. In another programming language, you would have to create the main input loop as part of your program.

The main input loop is executed as many times as there are lines of input. This loop does not execute until there is a line of input. It terminates when there is no more input to be read.

awk allows you to write two special routines that can be executed before any input is read and after all input is read. These are the procedures associated with the BEGIN and END rules, respectively. In other words, you can do some pre-processing before the main input loop is ever executed using the BEGIN procedure, and you can do some post-processing with the END procedure after the main input loop has been terminated. The BEGIN and END procedures are optional and they do not need to be defined.

You can think of an **awk** script as having potentially three major parts: what happens before, during, and after processing the input. The "what happens during processing" part is where most of the work gets done. Inside the main loop, your instructions are written as a series of pattern/action procedures. A pattern is a rule for testing the input line to determine whether or not the action should be applied to it. The actions can be quite complex, consisting of statements, functions, and expressions.

Appendix E. VMS to UNIX Command Reference

This appendix is intended as a convenient UNIX command reference for the migrating VMS user. The UNIX equivalents of commonly used VMS commands are listed in tabular format. No information on syntax, options, or arguments is presented here, however some of the listed commands are described elsewhere in *UNIX at Fermilab.*

Commands which are Fermilab-specific or AFS-specific are indicated as such.

E.1 UNIX Equivalents for Many VMS Commands

Symbols used in table:

These symbols are not part of any UNIX command with which they appear.

* for systems running AFS, the command may not work as documented

** your system may be configured to use an alternate Fermilab command to perform this action

\+ AFS-specific commands

\# local Fermilab commands

Actions	VMS	UNIX
Actions on files and directories		
create directory	CREATE/DIR	`mkdir`
change working directory	SET DEFAULT	`cd`
display working directory	SHOW DEFAULT	`pwd`
display list of files	DIRECTORY DIRECTORY/FULL	`ls` `ls -l`

display list of subdirectories only		`ls -l|grep '^d'`
		or
		`find * -type d \ -prune -print`

display contents of file	TYPE	`cat`
display file with pauses	TYPE/PAGE	`more, less`

Actions	VMS	UNIX
display first few lines of file	TYPE/PAGE	**head**
display last few lines of file	TYPE/TAIL	**tail**
copy a single file	COPY	**cp**
copy all files oldfile.* to newfile.*	COPY OLDFILE.* NEWFILE.*	Create shell script; see section E.2.
find file	DIRECTORY	**find**
find file containing a specified expression or string	SEARCH	**grep** **egrep** **fgrep**
compare files	DIFF	**diff** **cmp**
rename file	RENAME	**mv**
rename all files oldfile.* to newfile.*	RENAME OLDFILE.* NEWFILE.*	Create shell script; see section E.2.
delete file or directory	DELETE	**rm** **rmdir**
merge files	COPY or APPEND	**cat**
three-way file merge	MERGE	**merge**
sort or merge files	SORT	**sort**
assign (link) files	ASSIGN DEFINE SET FILE/ENTER	**ln** **setenv**
deassign files (remove link)	DEASSIGN	**rm**
convert and copy file	EXCHANGE	**dd**
update last modified date of file	OPEN/APPEND	**touch**
print file	PRINT	**lpr** **flpr #**
format text file	RUNOFF	**nroff** **troff** **groff #**
octal, decimal, hex, or ascii dump of file	DUMP	**od**
change file protection	SET FILE/PROT	**chmod** *

Actions	VMS	UNIX
change file ownership	SET FILE/OWNER	**chown** * **chgrp** *
list ACL settings on an AFS directory		**fs listacl** +
set ACL settings on an AFS directory		**fs setacl** +
General Commands		
get help	HELP	**man** **apropos**
logout	LOGOUT	**logout** **exit**
change password	SET PASSWORD	**passwd** * **kpasswd** + **yppasswd**[a]
obtain AFS (Kerberos) authentication		**klog** +
discard AFS tokens		**unlog** +
display logged-in users	SHOW USERS	**who** **finger**
display environment settings	SHOW LOGICAL	**alias** **setenv** **printenv** **set**
display date and time	SHOW TIME	**date**
display free disk space	SHOW DEVICE	**df** **du**
display disk quota (On AFS, shows AFS volume quota; see section 7.5.)	SHOW QUOTA	**quota -v** **fs listquota** ***pathname*** +[b]
write to screen	WRITE SYS$OUTPUT	**echo**
stop process	STOP	**kill**
display print queue	SHOW QUEUE	**lpq** **flpq** #
display print entries	SHOW ENTRY	**lpq** **flpq** #
delete print entry	DELETE/ENTRY	**flpk** #

Actions	VMS	UNIX
display processes	SHOW SYSTEM	**ps**
tape archiver[c]	BACKUP	**tar**
change terminal settings	SET TERMINAL	**setenv LINES** **setenv COLUMNS** **stty** (for Solaris)
talk to another user	PHONE	**talk**
disable messages	SET NOBROADCAST	**mesg n**
connect to remote node	SET HOST	**telnet** **rlogin**
copy file to a remote node	COPY FTP	**rcp** **ftp**
Commands for Executables, Procedures, and Batch Jobs		
compile FORTRAN program	FORTRAN	**f77**
link FORTRAN program	LINK	**f77**
see the entry point names in a library	LIBRARY/LIST/NAMES	**odump -c**[d]
execute program	RUN	***prog_name***
debug program	RUN/DEBUG	**dbx**
create or manipulate library	LIBRARY	**ar**
execute a command procedure (shell script)	*@prog_name*	***prog_name***
submit a batch job	SUBMIT/QUEUE	**qsub** **fbatch_submit #**[e]
check batch queue	SHOW QUEUE	**qstat** **fbatch_q #**
stop batch job	DELETE/ENTRY	**qdel** **kill** **fbatch_cancel #**

a. for use with NIS (see section 2.7)
b. For normal Berkeley-style systems, the UNIX command **quota -v** indicates how much of your disk quota you have available. On an AFS system, the quota is based not on individual users, but rather on AFS volume. So, if you have a shared volume, it just tracks the quota of the entire volume. For more information, enter **man fs_listquota** (the underscore is required).
c. See section E.3 for information on VMS backup save-sets.
d. under IRIX and OSF1

e. **fbatch** is a FUE interface used with the **LSF** job scheduling product, described in Chapter 14.

E.2 Shell Scripts for Copying/Renaming Multiple Files

In the above table, the UNIX command equivalents for copying/renaming a set of files require short shell scripts. In this section, we give you the script formulas.

The most efficient way to copy all files `oldfile.*` to `newfile.*` in UNIX is to create a short shell script. Its contents are shown for both the Bourne and the C shell families.

sh, ksh, or **bash:**

```
for file in oldfile.*
do
   newname=`echo $file | sed -e 's/oldfile/newfile/'`
   mv $file $newname
done
```

csh or **tcsh:**

```
#!/usr/local/bin/tcsh
foreach file (oldfile.*)
   set newname=`echo $file | sed -e 's/oldfile/newfile/'`
   mv $file $newname
end
```

Note that the shell variable *newname* in each case is set to the output of the command that follows the equals sign (=). This entire command string must be enclosed in back quotes (`...`). The argument for the **`-e`** option in the **`sed`** command is enclosed in normal single quotes (apostrophes). **sed** is not covered in this manual; most texts discuss it, and the book *sed & awk* published by O'Reilly & Associates covers it thoroughly.

E.3 Unpacking VMS Backup Save-sets

If you made VMS backup save-sets, the UNIX utility **vmsbackup** is a tool that lets you unpack the save-set, converting the files to UNIX format. **vmsbackup** is a separate **UPS** product which must be installed first (see section 10.3.1). The default operation of the program is to go through an entire VMS-generated tape, extract each file, and write it to disk. Several options are provided; we refer you to the man pages for a discussion of them.

Once **vmsbackup** is installed on your machine, you need to run **`setup vmsbackup`** before using it.

You cannot make save-sets with this utility; you can only unpack *existing* save-sets.

Appendix F. mh and exmh Customization

This appendix contains information for further customizing the **mh** and/or **exmh** mail readers. Given the multitude of customizable features in these mail readers, we cannot provide you with a comprehensive treatment of the subject here. The information presented in Chapter 12 and this appendix should be sufficient for most users.

We refer you to the on-line help and to the O'Reilly book, *MH & xmh - Email for Users & Programmers* by Jerry Peek, for further information.

F.1 Forwarding and Notification

The `.forward` file is a general mail forwarding file, and is used no matter which mail package you choose to use. It tells the system what to do with your mail; whether to leave it in the system spool area, forward it to the spool area on a different node, forward it to a different system entirely, or (in some cases) deliver it to your preferred local mailbox automatically. It may also be used as a means of notifying you that mail has arrived.

Note that comment lines are generally not permitted in the `.forward` file. Some OS versions and versions of **sendmail** (an underlying, transparent internet mail forwarding tool) do allow comment lines in `.forward`. The comment lines must be preceded by a pound sign (#).

F.1.1 Forwarding Address

The first line of your `.forward` file tells the system where to put your mail. On your chosen mail node, it should look like:

```
\{username}@{node}.fnal.gov
```

where `{username}` is your username, and `{node}` is the name of the node where you plan on reading mail (e.g., `\joe@fsui02.fnal.gov`). The backslash prevents infinite loops in a "clustered" environment where more than one node sees the same `.forward` file in your login area.[1]

On all other systems, we recommend that you forward your mail to FNAL, the lab's central mail server. For these systems, the first line of each `.forward` file will look like:

```
{username}@fnal.gov
```

e.g., `joe@fnal.gov`. Note that this doesn't have the backslash or the node name. The backslash is not necessary if you are forwarding to a node that does not share the same login area. See sections 2.6 and 2.7 for a discussion of shared login areas, or what we call UNIX *clusters*.

---

1. If you choose to use *unattended autoincorporation*, which delivers your mail directly into your login area rather than leaving it in the system spool area, the first line of your `.forward` file will look significantly different from this example. Unattended autoincorporation is covered in section F.4.

F.1.2 Mail Notification

Another use of the `.forward` file is to allow you to customize the type of notification you receive when new mail arrives. Usually, the second line of your `.forward` file will control if and how you receive notification.

FUE provides minimal notification of incoming mail by default, however most people like to receive notification of individual messages as they come in. If you want to scan your new mail when you log in, include the commands **`inc`** and **`scan +inbox unseen`** to your `.login` or `.profile` file.

The use of the **MH** program **rcvtty** provides notification of new mail whether or not you have a mail reader active. To set this up, you need to edit your `.forward` file.

Basic Configuration

Even if you don't know what **biff/comsat** is, you will need to determine if it is running on your system. How do you tell? Follow these instructions and you should get one concise notification message per incoming mail message if it is not running. If you get two notifications, one of which is quite verbose, then **biff/comsat** is running, and you'll need to change your configuration as described below.

If you're on a cluster where some nodes run **biff/comsat** and some don't, set up notification according to your chosen mail node. This procedure assumes that **MH** is installed and configured on all nodes within the cluster.

First, you need to make sure you have set up the proper file access permission. Activate tty user execute permission by using the command:

```
% chmod u+x `tty`
```

where the quotes around **`tty`** are both single back quotes.

You can include this in your `.login` or `.profile` file to run it automatically on all sessions. Be sure to include it *after* the call to `fermi.login` or `fermi.profile` since these files set it differently. Or, if you prefer to have notification only in one window, just run this command in that window.

The second line of your `.forward` file should look like:

```
"| /usr/local/products/mh/current/lib/rcvtty -biff"
```

When mail arrives, **rcvtty** should send a notice containing the time, sender, subject, and message sample to all user tty sessions with proper file access permission in the default format:

```
time:  SENDERNAME   subject << first line of text (truncated)
```

If this type of notification is what you get, skip to "Customizing rcvtty Output", below. If instead you get a multiline notice, **biff/comsat** is running, and you'll have to make a couple of changes.

Some systems run **biff** with **comsat** for notification purposes. **biff/comsat** and **rcvtty** (without the `biff` option) can work together, however this yields double notification (both **comsat**'s verbose notice and **rcvtty**'s single line notice) because the `r-w-x` permissions (see section 6.6.1) need to be different for **biff/comsat** and **rcvtty**.

To get **rcvtty** to work properly on systems running **biff/comsat** follow these steps:

1) Enter **`biff n`** to deactivate **comsat** (best to include in `.login` or `.profile` file).

2) In `.forward`, replace:

```
"| /usr/local/products/mh/current/lib/rcvtty -biff"
```

with:

```
"| /usr/local/products/mh/current/lib/rcvtty"
```

3) Enter **`ls -lL `tty``** to see your default tty setting. If `o+w` (that is, if *write* permission for *other*; see section 6.6.1) is not set, use the command **`chmod o+w `tty`**` to set it (again, best to include in `.login` or `.profile` file).

Customizing rcvtty Output

You can reformat the output of **rcvtty** if you like. To do this you need to create a form file describing the syntax of the output you want to see, and you need to modify the **`rcvtty`** command. For example, if you wish to use the same format as `scan-form` (see section F.2.5) the form file used for incorporating or scanning mail, the line in `.forward` would look like this (shown in the format used when **biff/comsat** is *not* running):

```
"| /usr/local/products/mh/current/lib/rcvtty -biff -form scan-form"
```

F.2 Files Used to Customize mh and exmh

Your personal configuration for **MH** is primarily defined via the `.mh_profile` file. It defines, among other things, where mail is stored and what editor to use. Some additional files are available if you wish to customize further. These were listed in section 12.4.

Each **MH** mail composition command, (**`comp`**, **`repl`**, **`forw`** for **mh**; Send, Reply, Forward for **exmh**) uses its own template file to define the message header format:

- **`comp`** or Send uses the `components` file
- **`repl`** or Reply uses `replcomps`
- **`forw`** or Forward uses `forwcomps`

If you don't have these template files in your `$HOME/Mail` directory the system default files in the **MH** library are used.

Options for incorporating mail were discussed in Chapter 12. The `scan-form` and `inc-form` files provide the format of the display when you scan and incorporate mail. You can control how local delivery is performed by creating a `.maildelivery` file.

F.2.1 .mh_profile

It also allows you to specify options to **MH** commands. If you have run **`setup mh`** or **`setup exmh`**, you already have a simple one in your home directory. You can tailor your `.mh_profile` file to suit your taste. Following are some of the extra lines you can add:

`Path: Mail`	Directory for your **MH** transactions.
`Unseen-Sequence: unseen`	Name of the unseen sequence of messages.
`Draft-folder: drafts`	Name of the default folder for drafted, unsent mail.
`Editor: emacs`	Sets up **emacs** as editor for **`comp`**, **`repl`**, **`forw`**, **`dist`** (use any editor you want).[1]
`rmmproc: /bin/rm`	Causes mail to be truly deleted when requested (see section 12.3.8). **Recommended!**

1. **exmh** ignores any editor defined here (defining an editor is optional for **MH**).

`Aliasfile: mh_alias`	File containing **MH** aliases (distribution lists).
`Scan: -form scan-form`	Provides format of display when you scan mail.
`inc: -form inc-form`	Provides format of display when you incorporate mail.
`Signature: John Peoples`	Your signature. Appears in `From:` header field.

The `Path` is taken to be relative to your `$HOME` directory. The files/directories used for `Draft-folder`, `Aliasfile`, `Scan`, and `inc` are assumed to be relative to the directory specified by `Path`.

F.2.2 components

The default template file for composing (sending) a message is called `components` and looks like:

```
To:
cc:
Subject:
-------
```

You can add other fields to the header by making your own `components` file. For example, an enhanced `components` file may include the additional fields shown below:

```
Reply-to: {username}@fnal.gov
To:
cc:
Bcc:
Fcc: +yourfolder
Subject:
-------
```

The `Reply-to:` field should give your FNAL address. It is not needed if you want the reply to come to the node from which you are sending the message. The **`repl`** command in **MH** (and some other mail agents as well) will route replies to this address rather than the `From:` address.

The "blind carbon copy" `Bcc:` field can be used to send copies of the message to addresses that are not seen by the people specified in the `To:` and `cc:` fields. This is like forwarding a copy of the message to someone else after sending it, except that here you can do it at the time you send the original message. `Bcc:` can also use distribution lists or aliases.

You can send a copy to yourself by specifying a folder for your copies in the `Fcc:` field. *Fcc* stands for folder copy. This copy goes directly to the specified folder, and only to that folder.

F.2.3 replcomps

The `replcomps` file controls the header in replies. The default file is given below:

```
%(lit)%(formataddr %<{reply-to}%?{from}%?{sender}%?{return-path}%>)\
%<(nonnull)%(void(width))%(putaddr To: )\n%>\
%(lit)%(formataddr{to})%(formataddr{cc})%(formataddr(me))\
%<(nonnull)%(void(width))%(putaddr cc: )\n%>\
%<{fcc}Fcc: %{fcc}\n%>\
%<{subject}Subject: Re: %{subject}\n%>\
%<{date}In-reply-to: Your message of "\
%<(nodate{date})%{date}%|%(pretty{date})%>."%<{message-id}
                %{message-id}%>\n%>\
--------
```

The first two lines in the file build the `To:` address field for your reply draft. It takes the first available address by searching the `Reply to:`, `From:`, `Sender:` and `Return-path:` fields, in that order. Lines 3 and 4 build a `cc:` address field by taking all the `To:` and `cc:` addresses from the original message and including your address as well. It then builds an `Fcc:` field by testing a **`-fcc`** switch on the command line or in the `.mh_profile`. The `subject:` field is taken from the subject in the incoming mail preceded by `Re:`. The last three lines make up the field that says "*In-reply-to: Your message of ...*" with the date and message number at the end. The default reply header takes the form:

```
To:
cc:
Subject:
In-reply-to: Your message of "Day, date time"
             <message-id>
--------
```

You can customize this formatting by modifying your own `replcomps` file (note that no spaces are allowed after the backslash (\) on each line). You will most likely need to reference the O'Reilly **MH** text for this task.

As you now know, `replcomps` controls the format of the message header. You can also customize the format of the message body for replies via a file called `.mh_filter` in your home directory. The O'Reilly **MH** book describes the different possible ways of setting this up. The contents of the default `.mh_filter` is:

```
body:component="> ",compwidth=0
```

F.2.4 forwcomps

The `forwcomps` file is used as the template header for forwarded messages. This file may look very similar to the `components` file, in fact the default files are identical. If you create your own `components` file and you want to use it for both sending and forwarding, include the following line in your `.mh_profile` file:

```
forw: -form components
```

You would only need a separate `forwcomps` file if you wanted the headers on forwarded and original mail to be different.

F.2.5 scan-form and inc-form

These files provide the format of the display when you scan and incorporate mail. If you aren't satisfied with the default scan format, we recommend that you simply set up a `scan-form` file, and use it for both situations (set both `Scan:` and `inc:` to `scan-form` in `.mh_profile`). The `scan-form` file's format is also used by **rcvtty** to provide messages. An example of a simple `scan-form` file showing incoming mail only and set up in a VMS-style format is shown below:

```
%4(msg)\
%21(addr{from})\
%02(mday{date})-%3(month{date})-%4(year{date})\
%02(hour{date}):%02(min{date})\
%{subject}
```

This example produces output in the format:

```
msg# sender_address  date  time  subject
```

Some sample output follows:

```
 18 helpdesk              13-Nov-1995 16:30 1017699 Problem Report
 19 lauri                 14-Nov-1995 10:56 VY0124 damaged
 20 STUART@fnald.fnal.gov 14-Nov-1995 11:28 Problems with an fnaldb tape drive
 21 wolbers@fsui01.fnal.g 14-Nov-1995 15:31 Department Meeting
```

An slightly more complicated example is shown below. It allows display of both incoming and outgoing messages:

```
%4(msg)\
%<(cur)+%| %>\
%<{forwarded}F%| %>\
%<{replied}R%| %>\
%<(mymbox{from}) <- %19(friendly{to}) %| -> %20(addr{from})%> \
 %02(mday{date})-%03(month{date})-%4(year{date})\
 %02(hour{date}):%02(min{date})\
 %{subject}
```

This example produces output in the format:

```
msg# (arrow) (sender or recipient address)  date  time  subject
```

The arrow indicates whether the mail is from you (<-) or to you (->). If it is from you, the address field shows who you sent it to. Sample output looks like:

```
 258   -> bhat                    14-Oct-1995 12:15 batch job 341: <run_small> D
 260   <- joe@fnal                13-Oct-1995 17:24 Program and Working Groups -
 261   <- lhn@sld.slac.stanford.edu 13-Oct-1995 16:31 e+w- Workshop, November 16-1
 262   <- martha@fnal             13-Oct-1995 17:24 re: revised mail chapter
 263   -> STROVINK@d0sfa          16-Oct-1995 09:21 Re: FYI re directions in NN
```

F.2.6 .maildelivery

The `.maildelivery` file controls how local delivery is performed. This file can be quite simple, or very complex. Within the file you can configure options to:

- restrict notification to mail received only from certain senders
- control notification format and content

- specify further processing such as delivery to one or more specific folders
- start up certain processes depending on information in the mail message header

Note that if you use the **mh** command `inc` (at the shell prompt) instead of incorporating within **exmh**, your mail will not be sorted. It will be filed into your `inbox` folder. The `.maildelivery` file is only used when you 1) incorporate within **exmh** (described in section 12.4.3) and 2) check the Presort box in the Incorporate Mail window under Preferences. The `.maildelivery` file is also used with unattended autoincorporation (see section F.4).

Simple Case: Send All Incoming Mail to inbox Folder

If you simply want to send all incoming mail to your `inbox` folder, your `.maildelivery` file needs only to consist of the following one line:

```
* - ^ A "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

If you want to do something more sophisticated, continue reading this section.

File Format

Lines starting with `#` are comments. Fields are separated by one or more spaces. Each record in the file has the following fields in this order:

header field	the message header field that you're sorting on (e.g., `from`, `to`, `subject`)
match pattern	the text string found in the header field that you want to match (enclosed in quotes if it includes spaces). If the header field argument is `default` or `*`, this should be `-` (a dash).
action	what to do if pattern is matched. Set to `file` or `>` to send the matched message to a file. Set to `qpipe` or `^` to directly execute a command without invoking a Bourne shell. Set to `pipe` or `|` to invoke a Bourne shell to execute the command. If your command has special characters in it, you will need to use `pipe` or `|`, otherwise `qpipe` or `^` is more efficient.
result	a code indicating whether to set matched message to "delivered", or not. `A`: if field and pattern matched, action will be performed, and if successful, then set message to "delivered". `R`: same as A, but don't set message to "delivered" so that further entries in `.maildelivery` can act on it. `?`: perform the action if message not yet set to "delivered" by a previous entry. If action succeeds, set message to "delivered".
file or command	depending on action, this is either a file (if action says "send message to a file"), or a command (if action says "execute a command"). Always include full pathname.

Note that some **MH** systems expect this file to have the mode 644 (`u+rw, g+r, o+r`).

Examples

The following examples should provide you enough information to modify your `.maildelivery` file to suit your tastes. We'll start with an annotated version of the simple `.maildelivery` file discussed above in which all your incoming mail gets placed in `inbox`:

```
# execute rcvstore to put all as-yet-undelivered incoming mail into inbox (sets messages
# to "delivered")
#hdr  pattern  action  result  command
 *    -        ^       A       "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

Here is a slightly more complex example. The comments explain what is being done in each line:

```
# File mail with fsui in the To: line into file $HOME/fsui.log
To fsui file A fsui.log

# Pipe messages from daemon@fnck to the (local) program err-message-archive
From daemon@fnck pipe A err-message-archive

# Put anything with Sender: address "mh-users" into $HOME/mh.log if not yet filed
Sender mh-users file ? mh.log

# If the address is "uas-admin", send an acknowledgment copy back
addr uas-admin ^ A "/bin/resendR-r $(reply-to)"

# Destroy anything from daemon@fnck!
From daemon@fnck destroy A -

# For anything not matched yet, execute rcvstore to put into inbox folder
* - ^ ? "/usr/local/products/mh/current/lib/rcvstore +inbox"
```

If you plan to create a significantly different set of criteria, you may need to see the man pages for **slocal** or the O'Reilly **MH** text referenced at the start of this appendix.

F.3 Automatic Reply to Incoming Mail

There is a handy UNIX program called **vacation** that you can use to send an automatic reply to emails that arrive while you're away. To enable **vacation**, enter simply:

```
% vacation
```

```
This program can be used to answer your mail automatically when you go away on vacation.
You need to create a message file in
/afs/fnal.gov/files/home/room3/aheavey/.vacation.msg first.
Please use your editor (emacs) to edit this file.
```

The editor invoked is determined by either the *VISUAL* or *EDITOR* environment variable. The default otherwise is **vi**. The default message is displayed in the editor window for you to edit. When you finish, the system returns:

```
You have a message file in /afs/fnal.gov/files/home/room3/aheavey/.vacation.msg.
Would you like to see it?
```

If you answer **y**, the file contents gets typed to the screen, and you are given the opportunity to re-edit the file. If you choose not to re-edit it, the system goes directly to:

```
You have a .forward file in your home directory containing:
(file contents displayed here)
Would you like to remove it and disable the vacation feature?
```

Respond **n**. At this point the dialog ends, and you need to go to your `.forward` file and add the line:

```
\{username}, "|/usr/bin/vacation {username}"
```

where `{username}` is your username, for example:

```
\aheavey, "|/usr/bin/vacation aheavey"
```

Now the message in `.vacation.msg` will get sent in reply to all subsequent incoming email messages.

To disable **vacation** when you return, simply remove the above line from your `.forward` file[1].

Run **man vacation** to get the full documentation.

F.4 Unattended Autoincorporation

It is possible, though not recommended in a *clustered* environment (see section 2.7 about clusters), to use the `.forward` file to automatically deliver your mail from the system mail spool area directly into your mail area, whether or not you are reading mail, or whether or not you are even logged in! This is called *unattended autoincorporation*. It has a number of attractive features:

- since mail doesn't really stop in the system spool area, but rather is delivered directly to your `inbox` file, you are no longer restricted to reading and receiving mail on one particular node of a cluster
- reduces the startup time (and de-iconizing time) of **exmh** because it doesn't have to "catch up" on lots of spooled messages
- mail can be automatically filed to specified folders as it is being delivered (otherwise only possible for **exmh**, not for the line-mode **mh**)

However, unattended autoincorporation also has a number of drawbacks and caveats which make it a less attractive option:

- in a clustered environment where several separate nodes (hence several separate mail spool areas) see the same mail destination `inbox` file, file locking can become a serious problem. It can happen that a deadlock results in *no* mail being delivered while an overload of processes attempt to deliver mail in vain. This problem is especially serious for people who receive a lot of mail, and is exacerbated by the use of **rcvtty** (mail notification; see section F.1.2).
- **exmh** does not update the visual display as new mail gets autoincorporated. Therefore, while you may have saved the time of incorporating at startup, you will need to manually Rescan all Folders (under the More... button in the middle window) in order to see that you have new mail (and in which folders) when you are running **exmh**.

Because of the detrimental effect that unattended autoincorporation can have on a system, and the fact that when it breaks, mail is not delivered, **we do not recommend its use in most circumstances**. In particular, we believe that it should not be used in clustered environments. It is documented here for completeness only, and should not be taken as an endorsement.

F.4.1 In Standard UNIX Environment

To configure unattended autoincorporation on a node:

1) create a `.maildelivery` file describing your mail sorting preferences (see section F.2.6)

---

1. Some OS versions and **sendmail** versions support comments in `.forward`. You can test to see if yours does. If it does, you can leave the vacation line in the file, and just comment it out with a pound sign (#) when you don't need it.

2) remove the line of your `.forward` (generally the first line) which contains something similar to `\{username}@{node}.fnal.gov`

3) replace it with "`| /usr/local/products/mh/current/lib/slocal -user {username}`" (include the double quotes but not the braces)

where `{username}` is your login id.

F.4.2 In AFS Environment

Proper ACL settings (AFS directory permissions, discussed in section 7.6.2) must be made for autoincorporation in **MH**. The easiest way to set your mail folder ACLs is to run the utility **fixmailperms** from your `$HOME` directory. This allows the mail folders that you've designated for receiving mail to work with **slocal**, and it sets *all* your mail folders to be readable only by your account and the system administrator's. **fixmailperms** is part of the **UPS** product **mailtools**. To see a brief explanation, enter:

```
% fixmailperms -help
```

The command is entered as:

```
% fixmailperms [folder_1 folder_2 ... folder_n]
```

The folder specifications are taken as relative to your `Mail` subdirectory. You need to specify all the folders into which you plan to automatically incorporate mail, and only those folders.

If you do not specify a minimum of one folder when you execute **fixmailperms**, unattended autoincorporation will not work.

For example, if you have a simple setup in which all incoming mail gets sent to your `inbox` folder, you'd enter:

```
% fixmailperms inbox
```

If this is satisfactory to you, skip to the next section. If instead you choose to set your ACLs individually (and not set your folders private), these are the ACLs you will need:

`system:anyuser rl`	for `$HOME`
`system:anyuser l`	for `Mail`
`system:anyuser li`	for `Mail/{folder}`[1]

Go to the directory whose ACL you want to change, and use the command syntax:

```
% fs setacl . system:anyuser xy
```

where *xy* is **`rl`**, **`l`**, or **`li`**, depending on the directory. Run **`fs listacl`** to see the current setting.

1. This folder should be your *inbox* or equivalent, e.g., `Mail/inbox`. You may need to set this for several folders, depending on the complexity of your autoincorporation set up.

Appendix G. mh Command Reference

This appendix provides an alphabetical reference to the subset of **mh** commands discussed in Chapter 12.

Remember to use the man pages if you need help on any of these commands.

Command	Description
% comp [-e *editor*]	Compose a message. This brings up a **vi** (or specified editor) session. Then save the message file, and exit the editor (see box below for more information).

Emergency **vi** exit: hit the Escape key (or <Ctrl-[>) and then enter:

:x	to save and exit, or
:q!	to quit without saving

At the `What Now?` prompt, you can enter a carriage return to get a list of options. Normally you will enter one of these options:

send	to send the message
quit	to save the message in `Mail/drafts` but not send it
quit -delete	to quit and not save

To send a message in which you want to include a file, compose the message, and use the "include file" command for **vi**:

:r *filename*

% folder +*new_folder_name*	Create a new folder.
% forw [*n*]	Forward the current message (or the specified message).
% inc	Incorporate new incoming message(s).
% next	Display the next message.
% pick -help	Display a list of pick options.
% pick *options*	Select messages by content, date, sender, recipient, etc.
% prev	Display the previous message.
% refile +*folder* [*n m* ...]	Move the current message (or specified messages) from the current folder to a different folder.
% repl [*n*]	Reply to the current (or specified) message. By default, everyone who got the original message gets a copy (the sender, plus everyone in `To:` and `cc:`). Respond to the `What Now?` prompt.
% repl_inc [*n*]	Like **repl**, but include the original message. Respond to the `What Now?` prompt.

% rmf ***+folder***	Delete the specified folder.
% rmm [*n m ...*]	Delete the current (or specified) message(s).
% scan [*+folder_name*]	Display message headers from current folder (or ***folder_name***, and set this folder to current).
% send ***file***	Send ***file*** to all the destinations defined, namely `To:` at a minimum, and `cc:`, if it's filled it in. (If you change your header to include `Bcc:` (blind copy), `Fcc:` (folder copy), it will send it to addresses in these categories as well.)
% setup mh	Setup the **mh** product.
% show ***+folder_name***	Display the current message in ***folder_name*** and set this folder to current.
% show [*n*]	Display the current (or specified) message.
% show ***n*** **| flpr -q** ***queue***	Print message number ***n*** on ***queue***.
% show ***n*** **>** ***filename***	Extract current (or specified) message into a file.
% show -showproc mhl ***n*** **| flpr -q** ***queue***	Print and format the message a little differently so that it's easier to read (and nicer for printing), and pipe it to **more**. Then send it to the printer.
% show -showproc pr ***m n*** **... | flpr -q** ***queue***	Print messages and separate them onto successive pages. Then send it to the printer.

Appendix H. Mail Conversion from VMS

This appendix is intended to guide you through a mail conversion process from VMS to the UNIX **MH** (Message Handling) system. Two options for conversion are presented.

See the introductory remarks to Chapter 12 before continuing.

H.1 Preparation for Conversion

Moving your VMS mail folders to UNIX is a process initiated from the UNIX system. It uses **ftp** (see section 13.1.1) or the **`rcp`** and **`rsh`** commands (see sections 13.1.2 and 13.1.3). Therefore, before you start, you need to set up two files:

- on your VMS machine (the remote node for this operation), a file named `.RHOSTS` that includes records for the UNIX node you're using as your mail destination. Section 13.1.3 describes the contents of this file, appropriate for both UNIX and VMS.
- `.mh_profile` on your UNIX mail node. To create this automatically, run **`setup mh`** or **`setup exmh`**. The file will contain the line `Path: Mail`, and that's all you need.

Make sure that v2_2 of **mailtools** is current. Enter:

```
% ups list -a mailtools
```

to see which version is current. If it is not v2_2 or higher, select the right version explicitly by typing:

```
% setup mailtools v2_2
```

If you are moving from FNALV, you should encounter no disk quota problem. The `USR$SCRATCH` area on FNALV is used to hold temporary files. If that is not set up on your VMS machine, you need to ensure that you have enough disk space on your UNIX system and enough disk quota on the VMS node to hold duplicate copies of all of your mail. For VMS, we suggest the commands:

```
$ SHOW QUOTA
```

to see how much space you have available, and

```
$ DIR/SIZE=ALL/GRAND *.MAI;*
```

(from the directory containing your mail) to see how much space you'll need.

For the UNIX side, as usual, it's not so cut and dried. You may be able to use **`quota -v`** or **`df`** to provide you the information; check the man pages on your system. On some systems (for instance FNALU running AFS), individual quotas may not be established.

H.2 Choosing the Process to Use

Now you're ready to convert your VMS mail folders to UNIX **MH** mail folders. You can use a "semi-automatic" process, or a totally automatic one.

For either process, the mail messages will be read and extracted into individual filenames in folders for use in **MH** or **exmh**. The filenames are incremented numbers starting at one (1), or the next highest available number for an existing **MH** folder. The **MH** folder name will be the same as the name of the mail folder on the VMS node (unless you rename it during a semi-automatic transfer); e.g., folder `SMITH` will become `/Mail/smith`.

The script used in both processes is `fvms2mh`. For full documentation on the **`fvms2mh`** command enter:

```
% fvms2mh -h |less
```

As mentioned above for those of you for whom `USR$SCRATCH:[username]` is not available or lacks sufficient quota, you need to have enough space on VMS for duplicate copies of your mail while you're doing the conversion. If you use the automatic method, you cannot use any scratch space (there is no "standard" way of assigning scratch space, therefore the automatic method has no way of knowing where it is). However, if you use the semi-automatic method, you can use scratch space for the EXTRACT/ALL second copy, the copy you bring over to your UNIX node.

H.3 Using the Semi-Automatic Process

This process won't take long unless you have many mail folders on VMS to convert. You can see and control all the steps using this method. Perform the following the steps:

1) Log in to the VMS node where your mail is kept.

2) For each folder in your MAIL file (and for each MAIL file that you maintain), enter:

```
MAIL> SET FOLDER foldername

MAIL> EXTRACT/ALL tempfile
```

3) Log in to the UNIX node where you intend to store your mail and use **ftp** or **rcp** to copy over to your `$HOME` directory the ***tempfile*** containing the extracted VMS mail messages. Sections 13.1.1 and 13.1.2 describe these file transfer utilities.

Do not create a `$HOME/Mail` directory and copy files directly there! Step 4 takes care of this.

4) Now you're ready to convert the copied mail files to **MH** mail folders. If for some reason you haven't run either **`setup mh`** or **`setup exmh`** on UNIX, do so now. Then, for each ***tempfile*** you copied over, enter:

```
% fvms2mh tempfile
```

H.4 Using the Automatic Process

This process simply automates all the steps of the semi-automatic process.

The `fvms2mh` script invoked in its simplest form:

```
% fvms2mh -vms vmsnode
```

performs the "EXTRACT/ALL" command on each of the VMS mail folders in your `MAIL.MAI` file in your default mail directory. Each folder is extracted into a separate file on the VMS node. Each file is then copied to the UNIX node and converted into **MH** format.

Note that ***vmsnode*** must be an individual node, not a cluster alias.

If you have other mail files, you'll need to convert them separately. For example, if all of your mail files are in the same (default) mail directory on ***vmsnode***, you can enter the command as:

```
% fvms2mh -vms vmsnode -file "\*"
```

We use the UNIX quoting syntax here to pass the asterisk wildcard to VMS so that all mail files are handled.

If your login id on VMS is different from your UNIX id, you'll need to add the `-l` switch:

```
% fvms2mh -vms vmsnode -l vms_login_id
```

Appendix I. Programming Examples

This appendix contains examples of programs that illustrate information presented in Chapters 16 and 17.

I.1 Interfacing C and FORTRAN

The following C/FORTRAN program illustrates some of the points discussed in section 16.9. Our executable is called `cfort`. We assume use of the **make** utility and Makefiles in this example. These are described in chapter 17.

The Makefile:

```
SHELL=/bin/sh
cfort : cf.o fc.o ; f77 -o cfort cf.o fc.o
fc.o  : fc.f      ; f77 -c fc.f
cf.o  : cf.c      ; cc  -c cf.c
```

The FORTRAN routine:

```
      PROGRAM CFORT

*     Test C/FORTRAN mixed programming techniques

*     Place COMMON variables in order REAL*8 , REAL/INTEGER , INTEGER*16 , BYTE
        COMMON /MIXED/ DCOMM, FCOMM, ICOMM
        REAL*8         DCOMM
        REAL           FCOMM
        INTEGER        ICOMM(2)

*     Arguments used in ICF function call
        INTEGER        INTE(2)
        REAL           FLO
        CHARACTER*10 STR

*     Initialize

        DCOMM    = 1.
        FCOMM    = 2.
        ICOMM(1) = 3
        ICOMM(2) = 30
        INTE(1)  = 4
        INTE(2)  = 40
        FLO      = 5.
        STR      = 'From f77'

C     Write initial values

        WRITE (*,*) ' '
        WRITE (*,*) ' / / / From FORTRAN / / / '
        WRITE (*,*) ' DCOMM  = ', DCOMM
        WRITE (*,*)' ' FCOMM ' = ', FCOMM
```

```
      WRITE (*,*) ' ICOMM  = ', ICOMM
      WRITE (*,*) ' INTE   = ', INTE
      WRITE (*,*) ' FLO    = ', FLO
      WRITE (*,*) ' STR    = ', STR
      WRITE (*,*) ' '

C     Use C function icf_, which prints, changes, and reprints COMMON and
C     Argument values

      IRET = ICF ( INTE , FLO , STR )

C     Write values after call to ICF
      WRITE (*,*) ' / / / Back in FORTRAN / / / '
      WRITE (*,*) ' DCOMM  = ', DCOMM
      WRITE (*,*) ' FCOMM  = ', FCOMM
      WRITE (*,*) ' ICOMM  = ', ICOMM
      WRITE (*,*) ' INTE   = ', INTE
      WRITE (*,*) ' FLO    = ', FLO
      WRITE (*,*) ' STR    = ', STR
      WRITE (*,*) ' IRET   = ', IRET
      WRITE (*,*) ' '

      STOP
      END
```

The C Routine (`icf_`):

```
            /*  Test of mixed C/FORTRAN features    */

int icf_( int* inte , float* flo , char* str , int lenstr ) {
extern struct {
    double dcomm ;
    float  fcomm ;
    int    icomm[2] ;
    } mixed_ ;

printf ( "  / / / Into C / / /\n\
  DCOMM  = %f\n\
  FCOMM  = %f\n\
  ICOMM  = %d %d\n\
  INTE   = %d %d\n\
  FLO    = %f\n\
  STR    = '%.10s'\n\
  LENSTR = %d\n\n"
  ,  mixed_.dcomm , mixed_.fcomm , mixed_.icomm[0] , mixed_.icomm[1]
  , *inte , *(inte+1) , *flo , str , lenstr ) ;

mixed_.dcomm    = 10000. ;
mixed_.fcomm    = 20000. ;
mixed_.icomm[0] = 30000  ;
mixed_.icomm[1] = 330000 ;
*inte           = 40000  ;
*(inte+1)       = 440000 ;
*flo            = 50000. ;

strcpy ( str , "Out of C" ) ;

printf ( "  / / / Out of C / / /\n\
  DCOMM  = %f\n\
  FCOMM  = %f\n\
  ICOMM  = %d %d\n\
  INTE   = %d\n\
  FLO    = %f\n\
  STR    = '%.10s'\n\n"
```

```
    , mixed_.dcomm , mixed_.fcomm , mixed_.icomm[0] , mixed_.icomm[1]
    , *inte , *flo , str ) ;

return ( lenstr ) ;

}
```

Program Execution (see section 16.10):

```
/ / / From FORTRAN / / /
DCOMM  =    1.000000000000000
FCOMM  =    2.000000
ICOMM  =             3          30
INTE   =             4          40
FLO    =    5.000000
STR    = From f77

/ / / Into C / / /
DCOMM  = 1.000000
FCOMM  = 2.000000
ICOMM  = 3 30
INTE   = 4 40
FLO    = 5.000000
STR    = 'From f77  '
LENSTR = 10

/ / / Out of C / / /
DCOMM  = 10000.000000
FCOMM  = 20000.000000
ICOMM  = 30000 330000
INTE   = 40000
FLO    = 50000.000000
STR    = 'Out of C'

/ / / Back in FORTRAN / / /
DCOMM  =   10000.00000000000
FCOMM  =    20000.00
ICOMM  =        30000      330000
INTE   =        40000      440000
FLO    =    50000.00
STR    = Out of C
IRET   =           10
```

I.2 Makefiles and the make Process

I.2.1 A Simple make Process

Our "Hello World" example is quite simple, and illustrates many of the features we have discussed in Chapter 17. First we'll run a file listing to see the files that are involved:

```
-rw-r--r--   1 aheavey  g020         515 Dec 20 16:27 Makefile
-rw-r--r--   1 aheavey  g020          91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020          88 Dec 20 16:27 model2.f
```

The contents of the FORTRAN source files that will be used as required files follows:

```
<fsui01> cat mcprog.f
      WRITE (*,*) ' Hello world '
      CALL MODEL1
      CALL MODEL2
      STOP
```

```
      END

<fsui01> cat model1.f
      SUBROUTINE MODEL1
      WRITE (*,*) ' Hello from MODEL 1 '
      RETURN
      END

<fsui01> cat model2.f
      SUBROUTINE MODEL2
      WRITE (*,*) ' Hello from MODEL 2 '
      RETURN
      END
```

Our Makefile illustrates the use of macro definitions, iterative target defintions, the use of a library in addition to other required files, comment lines, and macro expansion (described in the man pages). It provides two levels of "housekeeping" at the end, *clean* to remove just the "stray" files, and *clobber*, which gets rid of all resulting files thus allowing you to run **make** again under the same initial conditions.

```
<fsui01> cat Makefile
# use /bin/sh within make
SHELL = /bin/sh

#   Standard FORTRAN command and flags
F77    = f77
FFLAGS = -O2

# Build the 'mcprog' program from mcprog.f and libmc.a
mcprog   : mcprog.f   libmc.a ; $(F77) $(FFLAGS) -o mcprog mcprog.f libmc.a

# Move any missing modules into libmc.a
libmc.a  : model1.o  model2.o ; ar -r libmc.a $(?)

# Compile missing or out-of-date modules
model1.o : model1.f ;     $(F77) $(FFLAGS) -c $(?)

model2.o : model2.f ;     $(F77) $(FFLAGS) -c $(?)

clean    : ; rm -f *.o

clobber : ; rm -f *.o ; rm -f mcprog ; rm -f libmc.a
```

Run **make** without specifying a target, and it will use the first target it encounters, `mcprog`. Notice that **make** displays its progress on the screen, creating the library `libmc.a` prior to building the executable `mcprog`:

```
<fsui01> make
f77 -O2 -c model1.f model1.f
model1.f:
        model1:
model1.f:
        model1:
f77 -O2 -c model2.f model2.f
model2.f:
        model2:
model2.f:
        model2:
ar -r libmc.a model1.o model2.o
ar: creating libmc.a
f77 -O2 -o mcprog mcprog.f libmc.a
mcprog.f:
 MAIN:
```

Run another file listing to see that our target and some intermediate files were created:

```
-rwxr-xr-x   1 aheavey  g020      13452 Dec 20 16:34 mcprog*
-rw-r--r--   1 aheavey  g020       2712 Dec 20 16:34 libmc.a
-rw-r--r--   1 aheavey  g020       1248 Dec 20 16:34 model2.o
-rw-r--r--   1 aheavey  g020       1248 Dec 20 16:34 model1.o
-rw-r--r--   1 aheavey  g020         91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model2.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020        515 Dec 20 16:27 Makefile
```

Run the executable, `mcprog`:

```
<fsui01> mcprog
  Hello world
  Hello from MODEL 1
  Hello from MODEL 2
```

Run **`make clean`** to remove the object files:

```
<fsui01> make clean
rm -f *.o
```

A directory listing shows the object files were deleted:

```
-rwxr-xr-x   1 aheavey  g020      13452 Dec 20 16:34 mcprog*
-rw-r--r--   1 aheavey  g020       2712 Dec 20 16:34 libmc.a
-rw-r--r--   1 aheavey  g020         91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model2.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020        515 Dec 20 16:27 Makefile
```

Finally, run **`make clobber`** to remove all the created files:

```
<fsui01> make clobber
rm -f *.o ; rm -f mcprog ; rm -f libmc.a
```

... and check the final directory listing, which should match the original one:

```
-rw-r--r--   1 aheavey  g020        515 Dec 20 16:27 Makefile
-rw-r--r--   1 aheavey  g020         91 Dec 20 16:27 mcprog.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model1.f
-rw-r--r--   1 aheavey  g020         88 Dec 20 16:27 model2.f
```

I.2.2 A Physics Makefile

The following is an example physics Makefile with FORTRAN and C sources.

```
###############
#   MACROS    #
###############

# Always force the shell to be /bin/sh, for portability

SHELL=/bin/sh

# some variables come from the environment existing when make is run

#    CRNLIB       (from 'setup cern')
#    HISTO_DIR    (from 'setup histo')
#    MCFAST_DIR   (a group library from 'source ~bphyslib/mcfast/mcenv')
#    MCFIO_DIR    (a group library from 'source ~bphyslib/mcfast/mcenv')

# macros for directories containing INCLUDE files:

INC1 = $(MCFAST_DIR)/inc/event
```

```
INC2 = $(STDHEP_INC)

# macro for MCFAST object library directory

LIB = $(MCFAST_DIR)/lib.IRIX

# macro listing all USER object routines

OBJS  = \
  usr_analysis.o \
  usr_before_trigger.o \
  assign_off_mass.o \
  costh.o \
  fin_init.o \
  find_b_psiks.o

# macros for object libraries from products HISTO and CERN

HISTLIB = $(HISTO_DIR)/lib/libFHistoHB.a
PACKLIB = $(CRNLIB)/libpacklib.a
MATHLIB = $(CRNLIB)/libmathlib.a
KERNLIB = $(CRNLIB)/libkernlib.a

# macro for the SGI exception handler library

FP_EXCEPTION = /usr/lib/libfpe.a

# macro for all linked libraries

LOADLIBS= \
  $(LIB)/libgen.a      \
  $(LIB)/libdatafile.a \
  $(LIB)/libio.a       \
  $(LIB)/libutil.a     \
  $(MCFIO_DIR)/lib.IRIX/libFmcfio.a \
  $(HISTLIB) \
  $(PACKLIB) \
  $(MATHLIB) \
  $(KERNLIB) \
  $(FP_EXCEPTION)

# macros for compiler option flags

CFLAGS = -g
FFLAGS = -g -I$(INC1) -I$(INC2)

################
#   TARGETS    #
################

# f77 is used to run the link editor here.
# FORTRAN and C compilation is done via the rules defined below.

mcfast: $(OBJS) $(LOADLIBS); f77 -o mcfast $(OBJS) $(LOADLIBS)

################
#  SUFFIXES    #
################

# You must list all suffixes used in this Makefile's rules

.SUFFIXES : .c .F .f .o

################
#    RULES     #
```

```
################

# RULES for building *.o files from *.f or *.c

#     The $? internal macro expands to the list of *.f or *.c files which are
#     newer than their corresponding *.o file.

#     Macros CFLAGS and FFLAGS are defined above with compiler options

.f.o: ; f77 -c $(FFLAGS) $?

.c.o: ; cc  -c $(CFLAGS) $?
```

Index

Symbols

Hidden Files

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z