ANL/ESD/TM-145

Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS): Installation Manual

Argonne National Laboratory

Operated by The University of Chicago,
under Contract W-31-109-Eng-38, for the
United States Department of Energy

U.S. Army Corps of Engineers
Construction Engineering Research Laboratories (USACERL)
USACERL-98/128

US Army Corps of Engineers
Construction Engineering Research Laboratories

ANL/ESD/TM-145

Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS): Installation Manual

by

Z. Li, * P.J. Sydelko, * K.A. Majerus, * R.C. Sundell, and M.C. Vogt

Energy Systems Division, Center for Environmental Restoration Systems
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439

June 1998

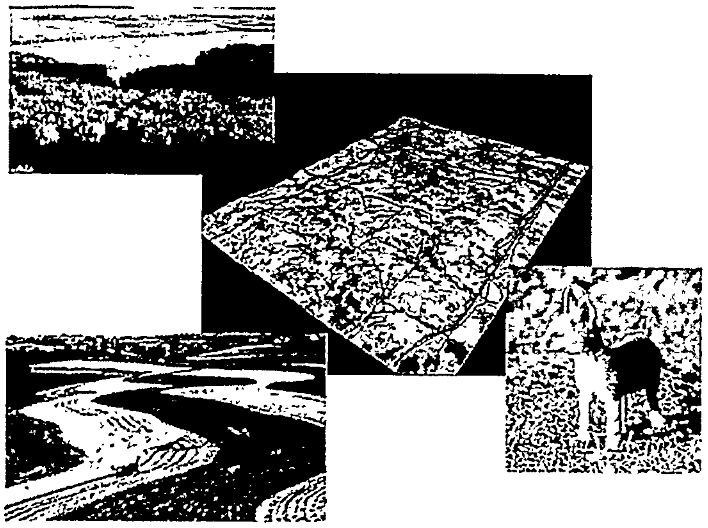
Work sponsored by Strategic Environmental Research and Development Program (SERDP)
Arlington, Virginia

DoD EPA DOE SERDP Strategic Environmental Research and Development Program
Improving Mission Readiness Through Environmental Research

*Li and Sydelko are affiliated with Argonne's Decision and Information Sciences Division; Majerus is affiliated with the U.S. Army Corps of Engineers, Construction Engineering Research Laboratories (USACERL), Champaign, Illinois.

About IDLAMS

As funding for conservation continues to decrease, many land managers are facing a challenge: to balance their goals of providing multiple land uses, while complying with natural resource regulations and minimizing negative environmental impacts. Actions to alleviate one problem often exacerbate others. A more integrated approach to land use planning and natural resource management is needed. To meet this need, Argonne National Laboratory and the U.S. Army Construction Engineering Research Laboratories have developed the computer-based Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS).



An integrated, dynamic modeling and decision-support system has been developed to provide a long-term strategic approach for land resource management.

IDLAMS supports the multiple objectives of sustaining natural resources, facilitating appropriate land use, and complying with regulations. This integrated, dynamic modeling approach is a promising tool that can help federal, state, and private organizations fulfill their land stewardship requirements while balancing multiple management objectives and supporting their primary missions.

All too often, decision-makers face critical land management decisions without sufficient information, such as thorough environmental data; information about other, competing objectives; and knowledge of land use requirements. IDLAMS, as developed for military use, consists of ecological, erosion, and training subroutines, along with advanced decision-support techniques, all linked with a core vegetation dynamics model that uses geographic information systems, remote sensing, and field inventory data. A user-friendly computer interface allows the land manager to operate this predictive, decision-support tool without the need for substantial computer or environmental modeling expertise. The key benefit of IDLAMS is that it can help land managers in three important ways: (1) strive toward multiple land use objectives using trade-off analysis, (2) evaluate the cost and economics of viable alternatives for managing land use, and (3) incorporate "what if" scenarios into decision-making. IDLAMS can also speed up responses to land-use management issues, improve environmental compliance, and help balance diverse land use needs.

Organizations interested in learning more about IDLAMS should contact one of the principal investigators listed on the following page.

A user-friendly computer interface allows the land manager to operate this predictive, decision-support tool without the need for substantial computer or environmental modeling expertise.

POINTS OF CONTACT

Through June 1998, Dr. Ronald C. Sundell served as project manager for this project. Dr. Sundell has left Argonne National Laboratory and currently holds a position at Northern Michigan University. Ms. Pamela J. Sydelko and Kimberly A. Majerus serve as points of contact for this project. For further information, please contact:

Ms. Pamela J. Sydelko
Argonne National Laboratory
Advanced Computer Applications Group
Decision and Information Sciences Division, Building 900
9700 South Cass Avenue
Argonne, Illinois 60439-4832

Phone: (630) 252-6727
E-mail: sydelkop@smtplink.dis.anl.gov

Ms. Kimberly A. Majerus
U.S. Army Corps of Engineers
Construction Engineering Research Laboratories
Natural Resource Assessment and Management Division
Land Management Laboratory
USACERL CECER LL-N
P.O. Box 9005
Champaign, Illinois 61826-9005

Phone: (217) 352-6511
E-mail: k-majerus@cecer.army.mil

PREFACE

The Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS) is a prototype, integrated land management technology developed through a joint effort between Argonne National Laboratory (ANL) and the U.S. Army Corps of Engineers Construction Engineering Research Laboratories (USACERL). Dr. Ronald C. Sundell, Ms. Pamela J. Sydelko, and Ms. Kimberly A. Majerus were the principal investigators (PIs) for this project. Dr. Zhian Li was the primary software developer. Dr. Jeffrey M. Keisler, Mr. Christopher M. Klaus, and Mr. Michael C. Vogt developed the decision analysis component of this project. It was developed with funding support from the Strategic Environmental Research and Development Program (SERDP), a land/environmental stewardship research program with participation from the U.S. Department of Defense (DoD), the U.S. Department of Energy (DOE), and the U.S. Environmental Protection Agency (EPA).

IDLAMS predicts land conditions (e.g., vegetation, wildlife habitats, and erosion status) by simulating changes in military land ecosystems for given training intensities and land management practices. It can be used by military land managers to help predict the future ecological condition for a given land use based on land management scenarios of various levels of training intensity. It also can be used as a tool to help land managers compare different land management practices and further determine a set of land management activities and prescriptions that best suit the needs of a specific military installation.

The IDLAMS project documentation consists of four reports:

1. IDLAMS Installation Manual*,
2. IDLAMS Programmer's Manual,
3. IDLAMS User's Manual, and
4. IDLAMS Final Report.

*This installation manual describes the system requirements, structure, and instructions for installing IDLAMS. More detailed information about IDLAMS development, methodology, or use can be found in the other three reports.

CONTENTS

FIGURES

ACKNOWLEDGMENTS

This report was funded by the Strategic Environmental Research and Development Program (SERDP). Special thanks go to Dr. Femi Ayorinde, SERDP Conservation Program Manager, for his assistance to and support of this project. In addition, special appreciation is also expressed to Craig Phillips, Dave Jones, Malcom Ponte, Herb Abel, and the Natural Resources staff at Ft. Riley, Kansas, for their help and cooperation in the development and testing of IDLAMS.

NOTATION

ANL	Argonne National Laboratory
ASCII	American Standard Code for Information Interchange
DoD	U.S. Department of Defense
DOE	U.S. Department of Energy
EPA	U.S. Environmental Protection Agency
ES	Energy Systems
GIS	Geographic information system
GRASS	Geographical Resources Analysis Support System
GUI	Graphical user interface
IDLAMS	Integrated Dynamic Landscape Analysis and Modeling System
MIM	Maneuver Impact Mile
PC	Personal computer
PI	Principal investigator
RUSLE	Revised Universal Soil Loss Equation
SERDP	Strategic Environmental Research and Development Program
Tcl/Tk	Tool command language/tool kits
USACERL	U.S. Army Corps of Engineers, Construction Engineering Research Laboratories

INTEGRATED DYNAMIC LANDSCAPE ANALYSIS AND MODELING SYSTEM (IDLAMS): INSTALLATION MANUAL

by

Z. Li, P.J. Sydelko, K.A. Majerus, R.C. Sundell, and M.C. Vogt

ABSTRACT

This manual provides instructions for the user on how to install the Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS). The system requirements, structure of the distribution package, and step-by-step installation instructions are provided. In addition, procedures for installing the GRASS geographic information system (GIS), Tcl/Tk graphical user interface (GUI), and GNUPLOT are provided in the appendixes to this report. The intended users of this manual are those responsible for installing, maintaining, and/or enhancing the IDLAMS system. It is assumed that the person installing the IDLAMS software package possesses a good working knowledge of C programming language(s), the UNIX operating system, and UNIX shell script languages. In addition, the installer should have an overall understanding of how to install GRASS GIS, Tcl/Tk, and GNUPLOT. The appendixes are meant to provide this overall understanding. Copies of the GRASS Programmer's Manual and GRASS User's Manual are essential and should be very helpful if modifications of the models are desired.

1 INTRODUCTION

The Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS) operates under the UNIX operating system and is built upon the Geographical Resources Analysis Support System (GRASS) geographic information system (GIS) (USACERL 1993). The vegetation dynamics model is the core of IDLAMS; its parts are either written in the C language or built with existing GRASS functions. External models or applications can be written in other languages or can exist in other GISs, provided that they accept ASCII array landcover maps as input and can export ASCII arrays back into IDLAMS. The entire system is integrated with a graphical user interface (GUI) by means of the Tcl/Tk programming language. To use this system, a UNIX workstation running with a SunOS 4.1.3 operating system or an IBM-compatible personal computer (PC) running with a UNIX operating system is required. Acceptable performance was achieved on a Sun SPARC Classic with 32 MB RAM, 2 GB disk space, and a 19-in., 8-bit video display.

IDLAMS currently consists of four major components:

1. The vegetation dynamics model,
2. Wildlife habitat submodels,
3. The soil erosion submodel, and
4. A scenario evaluation model.

The vegetation dynamics model is the core of the entire IDLAMS system. The wildlife habitat submodels and the soil erosion submodel all use the resultant vegetation cover map from the vegetation dynamics model as their landcover condition input to determine predicted wildlife habitat condition and soil erosion status. The scenario evaluation model evaluates the effectiveness of the selected land management practices on the basis of results from the vegetation dynamics model, wildlife submodels, and erosion submodel. The evaluations are made by using weighted multiple-attribute utility functions.

The entire system is integrated by means of an X-Windows GUI (written in Tcl/Tk programming language) to provide a user-friendly environment. Figure 1 presents a system diagram of the IDLAMS system.

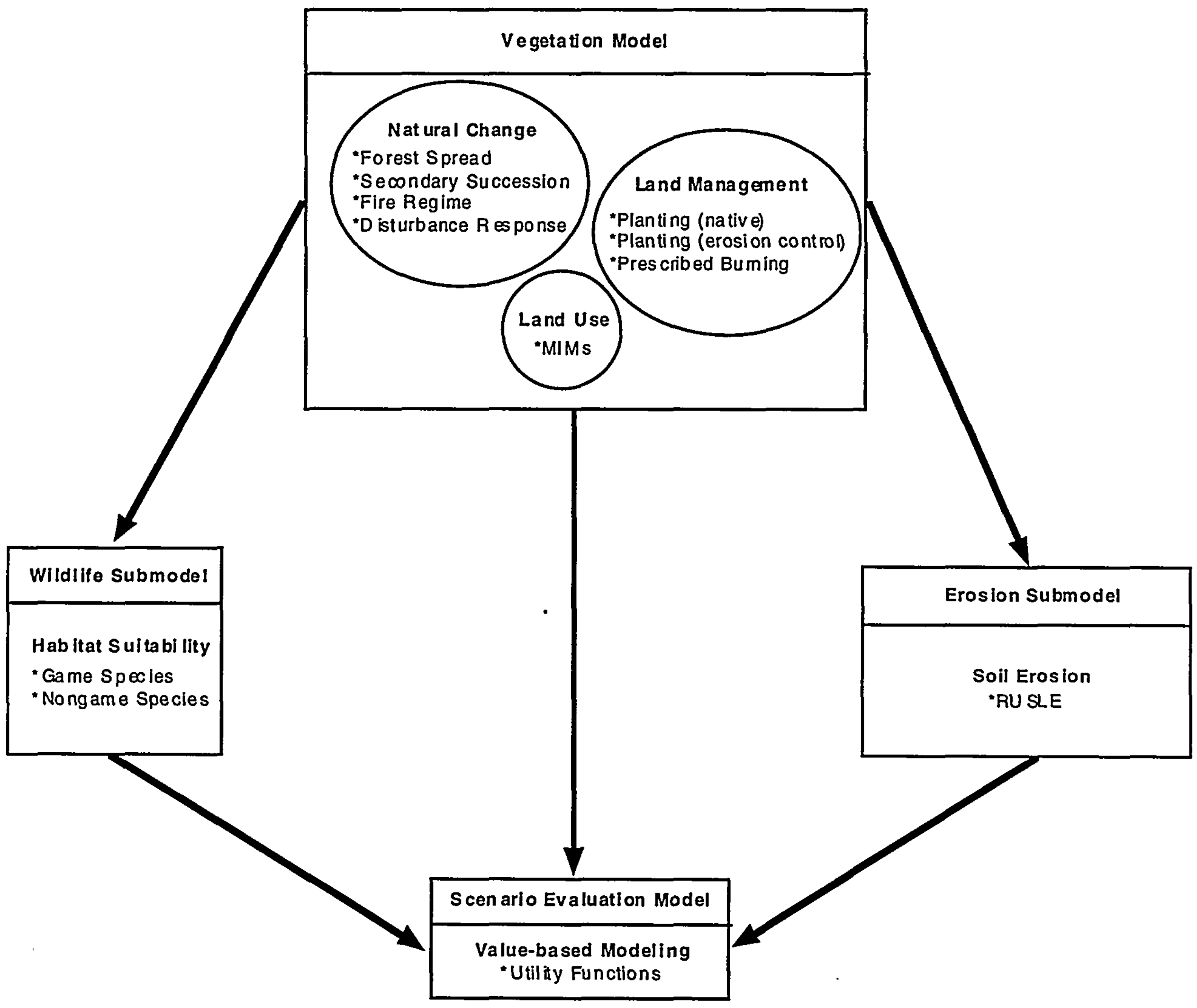
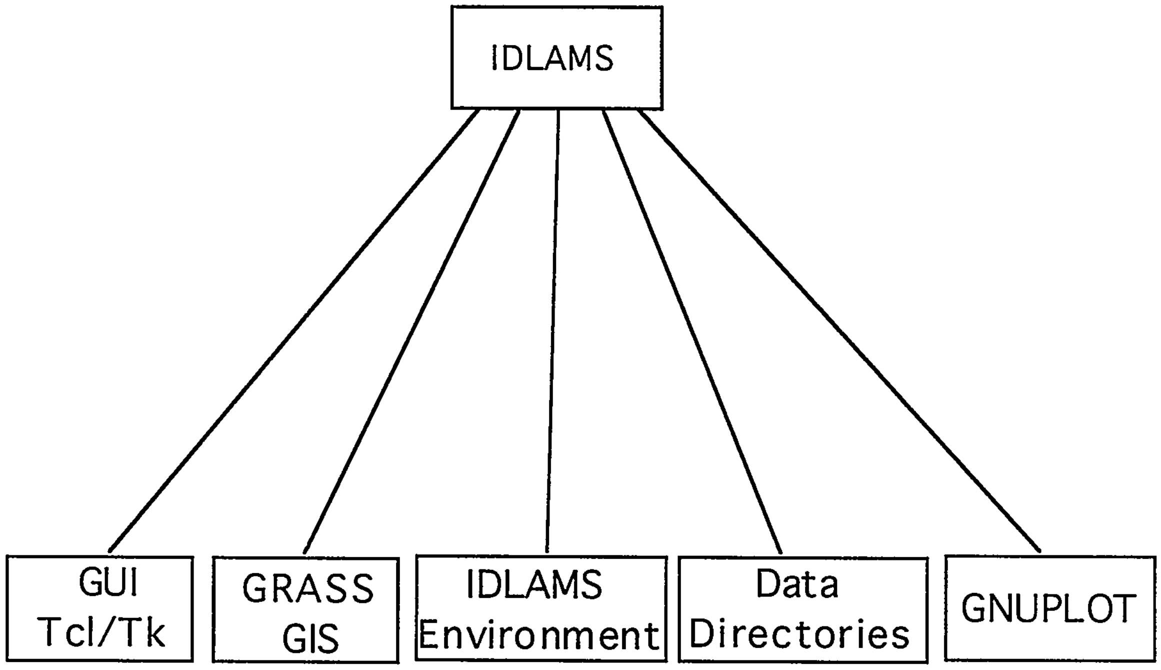


Figure 1 IDLAMS System Diagram

2 SYSTEM COMPONENTS AND REQUIREMENTS

The IDLAMS technology consists of links between various components, including software, data sets, files, and shell scripts. An overall representation of the major components of IDLAMS is provided in Figure 2.

The IDLAMS system consists of two major components, the GRASS GIS system and the IDLAMS models. Additional supporting packages include the Tcl/Tk ("expect", including "wish") language and GNUPLOT graphic utility. All of these packages are distributed together with IDLAMS for the user's convenience. However, users might not have to install these supporting packages if they are already up and running on the host machine. Details of the system components can be found in the IDLAMS Programmer's Manual.



GUI Tcl/Tk ————	Tool Command Language/Tool Kits
GRASS/GIS————	GRASS 4.1.3 with Vegetation Succession Model
IDLAMS Environment——	IDLAMS Integration Package, including: Wildlife Submodels Erosion Submodel IDLAMS Scenario Evaluation Model
Data Directories————	IDLAMS Input/Output Maps
GNUPLOT————	Graphic Software Package

Figure 2 IDLAMS Directory Hierarchical Diagram

IDLAMS was developed under the SunOS 4.1.3c UNIX operating system with an Openwindows X-window system environment. If the same operating system is used on the host machine, the compiled executable codes on the distribution media should run without need for recompiling. Further modifications may be necessary if IDLAMS is to be installed on machines with operating systems different from that stated above; in that case, all of the IDLAMS components, including the GRASS GIS system and Tcl/Tk, as well as GNUPLOT packages, must each be recompiled.

To load IDLAMS, a hard disk space of at least 100 megabytes is required. Additional disk space may be needed if larger data sets are to be stored and analyzed. It is recommended that one identify and determine the size of the required data sets and ensure that sufficient disk space is available.

3 STRUCTURE OF DISTRIBUTION PACKAGE

The IDLAMS software package consists of five major components. These five components are:

1. IDLAMS executable application, scripts, and support functions;
2. IDLAMS mapset of GIS GRASS data;
3. Floating point *GRASS 4.1* Floating Point Version Installation Manual with the IDLAMS Vegetation Model *r.veg.change*;
4. Tcl 7.3/Tk 3.6 (expect 5.7) Installation Manual; and
5. GNUPLOT -3.5 Installation Manual.

The distribution media contain the following files (file names are shown in italics) and contents.

- *README*: general instructions;
- *INSTALL*: shell script for loading IDLAMS;
- *SETUP*: shell script for building run-time parameters;
- *IDLAMS.tar.gz*: IDLAMS main Tcl/Tk source codes;
- *grassgis.tar.gz*: GRASS GIS system source code (with the IDLAMS Vegetation Model *r.veg.change*);
- *tcltk.tar.gz*: Tcl/Tk source code;
- *gnuplot.tar.gz*: GNUPLOT source code; and
- *IDLAMS mapset.tar.gz*: a demonstration mapset of GIS data.

The *README* file provides a brief general description of how to install the IDLAMS package. The *INSTALL* script is designed to automatically *uncompress* and un-*tar* the necessary components to the right directories, while the *SETUP* script has been developed to set up run-time parameters to minimize the directory structure dependency, allowing users to install the software anywhere in their local file system.

IDLAMS.tar.gz is the source code for the main program. The *grassgis.tar.gz* is the source code for the GRASS4.1 floating point GRASS GIS system with the Vegetation Succession Model, *r.veg.change*.

In addition, the IDLAMS system requires the Tcl/Tk programming language and the GNUPLOT graphic packages. For the user's convenience, these packages also are distributed with the IDLAMS system on the distribution media. They are *tcltk.tar.gz* and *gnuplot.tar.gz*, respectively. If the proper versions of these public domain software packages already exist on the machine, they do not need to be installed. However, it is recommended that IDLAMS be downloaded in its entirety. The last subdirectory, *IDLAMS mapset.tar.gz*, contains a demonstration mapset of GIS data. The user will need to test-run IDLAMS to verify that the system has been correctly installed.

4 INSTALLATION INSTRUCTIONS

The following eight steps are necessary to install the IDLAMS software package:

1. Create an IDLAMS user login and corresponding IDLAMS home directory.

2. Download the file from the media by using the UNIX *tar* command.

3. Run the *INSTALL* script. During installation, the user is prompted to provide installation locations for each of the components to be installed. After all of the questions are answered, the *INSTALL* script will *uncompress* and un-*tar* the necessary files to the designated directories.

4. Compile GRASS4.1. For details on how to compile GRASS4.1, please see the attached GRASS4.1 installation manual in Appendix A.

5. Compile the Tcl/Tk package (if this package has not already been loaded on the machine). Both the expect and wish components of Tcl/Tk are needed for IDLAMS. For details, please see the attached Tcl/Tk installation manual in Appendix B.

6. Compile GNUPLOT package (if this package has not already been loaded on the machine). For details, please see the attached GNUPLOT installation manual in Appendix C.

7. Edit the *.cshrc* or *.profile* file in IDLAMS *$HOME* directory to add Tcl/Tk, GRASS4.1, and GNUPLOT in the search path.

8. Run the *SETUP* script to establish the run-time parameters. If the *SETUP* script runs successfully, IDLAMS is now set up and ready to run. It is important to enter the full path names of the directories during setup. For example, if the *wish* command of Tcl/Tk resides in */usr/local/bin/tcltk/*, the user must enter it as */usr/local/bin/tcltk*.

Finally, the person responsible for installing the IDLAMS system is cautioned that in this first version of IDLAMS and no extensive validation check for the correctness of the user's input parameters has been made. The user should double-check the directory names he/she entered during installation and setup. If, when one is starting IDLAMS, an error message states that a parameter(s) is missing, the user should double-check the parameters entered during setup and rerun the *SETUP* script.

5 SUPPORTING DOCUMENTATION

For convenience, the following documents have been attached as appendixes to this installation manual:

A. GRASS4.1 Installation Manual (1993);

B. Tcl 7.3/Tk 3.6 (1995) and expect 5.7 (1994) Installation Manual; and

C. GNUPLOT 3.5 Installation Manual (1993).

However, for a greater level of detail, it is suggested that the person installing the system obtain all related supporting documentation.

6 REFERENCES

Ousterhout, J., 1995, Tcl/Tk Installation and Documentation Directory, University of California at Berkeley, ouster@cs.berkeley.edu.

USACERL: See U.S. Army Corps of Engineers, Construction Engineering Research Laboratories.

U.S. Army Corps of Engineers, Construction Engineering Research Laboratories (USACERL), 1993, Geographic Resources Analysis Support System (GRASS), version 4.1, User's Reference Manual, Champaign, Ill.

Williams, T. and C. Kelley, 1993, GNUPLOT, An Interactive Plotting Program, version 3.5, info-gnuplot-request@dartmouth.edu.

APPENDIX A

GRASS 4.1 FLOATING POINT VERSION INSTALLATION MANUAL

GRASS 4.1 INSTALLATION GUIDE

ABSTRACT

This document explains how to install GRASS 4.1. **Please read the instructions completely before beginning. There are changes in the installation process between GRASS 4.0 and GRASS 4.1.**

April 30, 1993

Table of Contents

GRASS 4.1 INSTALLATION GUIDE

1. SUPPORT

The Geographical Resource Analysis Support System (GRASS) is a public-domain product of the GRASS Inter-Agency Coordinating Committee. The 4.1 release is being coordinated by the Office of GRASS Integration (OGI) at the Construction Engineering Research Laboratories (CERL), Champaign, Illinois. GRASS is an integrated set of many programs designed to provide digitizing, image processing, map production, and geographical information system capabilities to the user. Authorship of the individual programs is noted in the *GRASS 4.1 User's Reference Manual.*

CERL maintains the GRASS Information Center at phone number 1-800-USA-CERL (or 217-352-6511), extension 220. Through this Center you may find out what firms and agencies currently distribute and support GRASS as well as which offer class instruction, data development, or related services.

2. TO COMPILE OR NOT TO COMPILE?

Your release tape may provide you with binary code compiled for a particular machine configuration. If this configuration (which is clearly marked on the medium's label) matches your machine set up, you will be able to run GRASS almost immediately after loading the files onto disk. The simple instructions that come with the distribution media should be followed and this entire document can safely be ignored for now.

These instructions are for sites that have received source code. Compilation of this software can be moderately to extremely difficult. You may attempt installation yourself or seek assistance through any of several commercial firms competent in GRASS installation and support. Refer to section 1 for finding assistance.

This installation document involves the compilation of computer source (human-readable) code into machine language. To be successful. you must be familiar with:

Your computer
UNIX
UNIX shell(s) (sh. csh)
An editor
A C language compiler
Makefiles
Tape handling and reading (using the **mt** & **cpio** commands)
Super-user operations

configurations:[1]

[1] See section 17. "Appendix." for notes regarding other computer configurations.

	COMPUTER SUN (4,386i)
Operating System	SunOS 4.1
C compiler version	any K&R compatible
Graphics software	Sunview, X-windows
Graphics Hardware	8-plane color

GRASS 4.1 has been compiled and runs on the following computer If you are using different configurations than those defined here, expect more than the usual share of difficulties. Even a "new and improved" compiler can cause problems. Installation of GRASS on systems using new (to GRASS) graphics monitors will experience the most difficult problems. The *GRASS 4.1 Programmer's Manual* is available to help the person responsible for porting GRASS to different hardware configurations.[2]

3. DATA IMPACTS

Existing GRASS data files (those created under previous versions of GRASS) can be used under GRASS 4.1 without conversion.

Please note that while there have been no significant changes to the vector, raster, site and imagery data files from 4.0 to 4.1, there is no guarantee that data created under 4.1 will be backwards compatible with 4.0.

Vector files and their support files as well as raster color files made under GRASS 4.1 (or 4.0) are not backwards compatible with GRASS 3.1 or earlier versions of GRASS.

4. PREPARATION

You must first decide which user will own the GRASS 4.1 files. In this document, we will assume that the user *grass* will own the GRASS 4.1 files.

GRASS 4.1 should be installed in new directories. DO NOT extract the 4.1 source code on top of any existing 4.0 source. If you already have a version of GRASS on your system and you want to install GRASS 4.1 in the same place, move the older version elsewhere.[3] Compiled binaries should go into a new place as well. OGI recommends that you keep the SRC and GISBASE directories separate for more flexibility; however, if you have a shortage of available disk space, you may elect to use the same directory for both.

[2] The *GRASS 4.1 Programmer's Manual* can be used as a reference for writing graphics drivers.

[3] A certain amount of savvy is required here. If the older version exists in the home directory for a user, and you rename the home directory, not only must you recreate the directory, but you must also restore the user's files as well (e.g., *.cshrc*, *.login*, *.mailrc*).

GRASS 4.1 source and programs can be installed anywhere in your computer system. This document will henceforth refer to the directory that houses the GRASS software as

```
    $GIS
```

You should also be careful about file permissions when installing some of the GRASS programs; in particular, GRASS installs the programs **gmake4.1** and **grass4.1** in a commonly accessible directory (defined when running the **setup** program). In most cases, the GRASS installation will not be performed directly by the super-user (in fact, the installation is often performed by the user *grass*); in such cases, take care to ensure that the directory where **gmake4.1** and grass4.1 are installed has permissions allowing the user installing GRASS to copy these programs into this directory. (You can do this by allowing only members of a particular user group to write to the directory and then making the installing user a member of that group; you can also accomplish this by making the directory world-writable temporarily, i.e., until the GRASS installation is completed. Note that you must set up the permissions before starting the installation process because the **setup** program will not let the installer select a destination directory that is not writable.)

Similarly, GRASS 4.1 data can be installed anywhere in your computer system. At this time you must determine where you would like to load the sample data. This document will henceforth refer to this data directory as

```
    $GISDBASE
```

You can use the UNIX shell variables to store the paths of the GRASS source directory and the GRASS data directory in the GIS and GISDBASE variables, respectively. For example, if you choose to place the software in */usr/grass4.1,* you might use one of the following commands:

```
  For CSH    setenv  GIS  /usr/grass4.1
  For SH     GIS= /usr/grass4.1; export  GIS
```

Similarly, if you choose to place the data in */usr/grass.data* you might use one of the following commands:

```
  For CSH    setenv  GISDBASE  /usr/grass.data
  For SH     GISDBASE= /usr/grass.data; export  GISDBASE
```

Create the software and data directories by issuing the following commands (you may have to **su** to *root* to create them – if you do, be sure to change the ownership to *grass*):

```
    mkdir  $GIS
    mkdir  $GISDBASE
    chown grass $GIS $GISDBASE
```

Note that the GIS and GISDBASE shell variables must be set as defined above.

Then log in as *grass*, set the GIS and GISDBASE variables as above, and change to the directory $GIS:

```
cd $GIS
```

5. DISTRIBUTION TAPE CONTENTS

The distribution media contain one or more files. (Check your label for verification.) The following table shows the contents of these files:

File	Contents	Size	
Number		Uncompiled	Compiled
1.	GRASS Source code	49.7 Mb	< 97.5 Mb
2.	Spearfish database	17.2 Mb	N/A
3.	Imagery database	30.4 Mb	N/A
4.	World database	2.3 Mb	N/A
5.	Related Source code	12.9 Mb	< 18 Mb

6. EXTRACTING SOURCE CODE AND SAMPLE DATABASE

Mount the distribution tape and then change to the directory $GIS. Follow the instructions in the table that represents your machine.[4] It is recommended that you extract the files as the user *grass*, not as *root*. If you do it as *root*, the files may be owned by a random or unknown user on your system.

The five files can be extracted from the distribution tape individually at any time. For example, it may be appropriate to extract the source code one day for compilation, extract the Spearfish sample database another day, and then unload the sample imagery data another day.

The device names used in the tables (e.g., "/dev/rmt0") are only examples; you must use the correct device names for your hardware configuration. Furthermore, the rewindable and non-rewindable names for the tape device must be used in the appropriate steps of loading the tape (e.g., you must substitute your rewindable tape device name wherever "/dev/rmt0" is given in the instructions, and the non-rewindable device name wherever "/dev/nrmt0" is listed).[5]

[4] The files are stored on the tape with *relative* path names so they will be extracted into and below the current directory. **Make sure you are in $GIS or $GISDBASE prior to issuing the commands necessary to extract the source code or sample database respectively.**

[5] For every physical tape drive, UNIX provides two device names, corresponding to a rewinding device and a non-rewinding device. When the rewinding device is used in an operation, UNIX will rewind the tape once the operation is completed: when the non-rewinding device is used, the tape will be left at the position reached at the end of the operation. As a result, two names can be used (with different meanings) to refer to

For both of the SUN tape formats as well as for the Intergraph formats, note that if you intend to unload all five files in the order that they are present on the tape, then you should be able to unload them without having to rewind and position the tape for each step – this will shorten the unloading process considerably.

SUN ½" tape	
load tape	
Unload Programs	
go to GRASS directory	cd $GIS
rewind tape	mt -f /dev/rmt0 rew
extract source	dd if= /dev/rmt0 ibs= 20480 |cpio -icdu
Unload Spearfish database	
go to DATA directory	cd $GISDBASE
rewind tape	mt -f /dev/rmt0 rew
position tape*	mt -f /dev/nrmt0 fsf 1
extract spearfish data	dd if= /dev/rmt0 ibs= 20480 |cpio -icdu
Unload Imagery database	
go to DATA directory	cd $GISDBASE
rewind tape	mt -f /dev/rmt0 rew
position tape*	mt -f /dev/nrmt0 fsf 2
extract imagery data	dd if= /dev/rmt0 ibs= 20480 |cpio -icdu
Unload World database	
go to DATA directory	cd $GISDBASE
rewind tape	mt -f /dev/rmt0 rew
position tape*	mt -f /dev/nrmt0 fsf 3
extract world data	dd if= /dev/rmt0 ibs= 20480 |cpio -icdu
Unload Related Source code	
go to GRASS directory	cd $GIS
rewind tape	mt -f /dev/rmt0 rew
position tape*	mt -f /dev/nrmt0 fsf 4
extract related src	dd if= /dev/rmt0 ibs= 20480 |cpio -icdu
*You must specify the **n** in the device name on these commands.	

the same physical tape drive.

SUN ¼" cartridge	
load tape	
Unload Programs	
go to GRASS directory rewind tape extract source	cd $GIS mt -f /dev/rst0 rew dd if= /dev/rst0 ibs= 65536 |cpio -icdu
Unload Spearfish database	
go to DATA directory rewind tape position tape* extract spearfish data	cd $GISDBASE mt -f /dev/rst0 rew mt -f /dev/nrst0 fsf 1 dd if= /dev/rst0 ibs= 65536 |cpio -icdu
Unload Imagery database	
go to DATA directory rewind tape position tape* extract imagery data	cd $GISDBASE mt -f /dev/rst0 rew mt -f /dev/nrst0 fsf 2 dd if= /dev/rst0 ibs= 65536 |cpio -icdu
Unload World database	
go to DATA directory rewind tape position tape* extract world data	cd $GISDBASE mt -f /dev/rst0 rew mt -f /dev/nrst0 fsf 3 dd if= /dev/rst0 ibs= 65536 |cpio -icdu
Unload Related Source code	
go to GRASS directory rewind tape position tape* extract related src	cd $GIS mt -f /dev/rst0 rew mt -f /dev/nrst0 fsf 4 dd if= /dev/rst0 ibs= 65536 |cpio -icdu
*You must specify the **n** in the device name on these commands.	

Intergraph ¼" cartridge and 8mm cartridge	
load tape	
Unload Programs	
go to GRASS directory rewind tape extract source	cd $GIS mt -f /dev/rmt/0m rew dd if= /dev/rmt/0m ibs= 20480 | cpio -icdu
Unload Spearfish database	
go to DATA directory rewind tape position tape* extract spearfish data	cd $GISDBASE mt -f /dev/rmt/0m rew mt -f /dev/rmt/0mn fsf 1 dd if= /dev/rmt/0m ibs= 20480 | cpio -icdu
Unload Imagery database	
go to DATA directory rewind tape position tape* extract imagery data	cd $GISDBASE mt -f /dev/rmt/0m rew mt -f /dev/rmt/0mn fsf 2 dd if= /dev/rmt/0m ibs= 20480 | cpio -icdu
Unload World database	
go to DATA directory rewind tape position tape* extract world data	cd $GISDBASE mt -f /dev/rmt/0m rew mt -f /dev/rmt/0mn fsf 3 dd if= /dev/rmt/0m ibs= 20480 | cpio -icdu
Unload Related Source code	
go to GRASS directory rewind tape position tape* extract related src	cd $GIS mt -f /dev/rmt/0m rew mt -f /dev/rmt/0mn fsf 4 dd if= /dev/rmt/0m ibs= 20480 | cpio -icdu
*You must specify the **n** in the device name on these commands.	

7. GRASS COMPILATION

If your site has several different machine architectures for which GRASS needs to be compiled, you have the option to mount a single, shared copy of the source code on the different machines (via NFS). You may then follow these instructions once for each machine to produce a separate set of object and executable files for each architecture; the compilation process will ensure that object and binary files for the different architectures are kept separate.

These instructions presume that you are familiar with UNIX, C, make, and shell scripting.

NOTE: These instructions and scripts have been used to compile GRASS on the following machine architecture:[6]

SUN 4 with SunOS 4.1.1

Please e-mail comments regarding the compiling of GRASS on other platforms or operating systems to:

[6] But see section 17 "APPENDIX" for information about other platforms.

```
grassbug@zorro.cecer.army.mil
```

At this point the GRASS source code has been loaded and is ready to be compiled. Most of the compilation steps take place within the directory $GIS/*src/CMD*. While logged in as *grass*, the owner of the GRASS code, do the following:

```
cd $GIS/src/CMD
ls
```

This directory contains scripts and files used to compile GRASS. By running scripts and changing lists of programs, you generate GRASS binaries for your system. The following is a partial list of the contents of this directory:

VERSION

Current version number and date of the GRASS release

generic/

System-independent files needed by the compilation process

GISGEN.sh	shell script that controls the entire compilation process
MAKELINKS.sh	shell script used to establish links to the user executable commands
gmake.sh	shell script that does compilations
make.head	make variables
make.tail	some additional make rules

head/

This directory is empty on the distribution tape, but will contain header file(s) for this site. Header files are created by running the *utils/setup* command.

lists/

Lists of programs to be compiled:

GRASS	standard GRASS programs
local	site-specific GRASS programs (you will create this file)
SARCH	architecture-dependent GRASS programs (you will create this file)
local.example	sample list of optional programs and drivers that can be compiled locally

next_step/

This directory is empty on the distribution tape. It will contain files that track how far along the compilation process is.

utils/

Contains the *setup* script and its related scripts and files.

7.1. COMPILATION STEPS OVERVIEW

(1) Generate files that contain site- and machine-specific information for the make program.

(2) Please check the Appendix for any alterations that may be required for your platforms.

(3) Edit files containing lists of local and machine-architecture-specific programs to be compiled (generally printer, digitizer, and graphics drivers).

(4) Run the GRASS compilation script.

(5) Run the GRASS program linking script.

(6) Edit device driver configuration files.

(7) Compile GRASS contributed programs.

(8) Compile GRASS related and hybrid programs.

7.2. COMPILATION SUPPORT FILES

Each machine and site needs to have GRASS compiled in ways that specify different:

- compilation and load flags
- system libraries
- installation directories
- default databases and locations

The shell script *utils/setup* assists you in defining many of the compilation options and definitions that will become part of every compile-time generated makefile. The *setup* script creates four files:

< header>
: A file containing system-dependent information used during compilation. The *setup* script will ask you for a name for this < header> file. It will be created in the *head/* directory. We suggest using the name of the architecture, similar to $ARCH.

gmake4.1
: This shell script does compilation (using the UNIX **make** command). It will use the < header> information during compilation. The script is placed into a directory (called UNIX_BIN) that you specify during the setup. This directory must be part of, or added to, your shell's PATH variable.

GISGEN.< header>
: This script compiles all of GRASS. It is created in the $GIS/*CMD* directory.

MAKELINKS.< header>
: This script links all user executables to a common program called **front.end**.

NOTE: For brevity, this document will refer to *gmake4.1* as *gmake*, *GISGEN*.< header> as *GISGEN*, and *MAKELINKS*.< header> as *MAKELINKS*.

NOTE: *GISGEN* assumes **/etc/mknod** is the command for making fifos. This is not true for all machines. Installers should modify *generic/GISGEN.sh* to replace the **/etc/mknod/** command with the one correct for their **/bin/mknod**).

7.3. COMPILATION SETUP

Run the *setup* script[7] and answer the questions it asks:

```
sh utils/setup
```

When *setup* is complete, it will ask you for the name of a file to store the information it gleaned about your system. This document refers to this file as < header> . You are encouraged to use the same name as $ARCH (the architecture name).

Examine the newly created < header> file in *head/* to make sure things are OK. A brief description for each defined variable follows.[8]

CC	The name of the C compiler.
ARCH	Name identifying the architecture of the machine on which you are compiling GRASS.
GISBASE	Directory where compiled GRASS will be installed.
UNIX_BIN	Local directory where the GRASS *grass4.1* entry program and *gmake4.1* compilation script will be installed. This directory should be writable by the user building GRASS.
DEFAULT_DATABASE	Directory where local GRASS databases are contained.
DEFAULT_LOCATION	GRASS database that first-time users get as a default.
COMPILE_FLAGS	Compilation flags.
LDFLAGS	Load flags.
TERMLIB	System library supporting low-level cursor movement.
CURSES	System library that supports screen cursor control.
MATHLIB	System math library.
LIBRULE	Method for archiving and randomizing libraries.
USE_TERMIO	Flag to use the termio library if available.
USE_MTIO	Flag to use the mtio library if available.
DIGITFLAGS	Flags to set owner and priority of the v.digit program.
XCFLAGS	Flags for X11 compilation.
XLDFLAGS	Loader flags for X11 programs.

[7] If you are using Control Data EP/IX (or RiscOS), please see the notes regarding Control Data Hardware in section 17, "Appendix."

[8] See section 17 "APPENDIX" for sample < header> files that have been created at CERL for sun4 and other machines.

XINCPATH	Directory for X11 include files.
XMINCPATH	Directory for Motif include files.
XLIBPATH	Directory for X11 library.
XTLIBPATH	Directory for Xt library.
XMLIBPATH	Directory for Motif libraries.
XEXTRALIBS	Platform specific X11 libraries.

7.4. EDIT LISTS

Next, you must edit files containing lists of site- and machine-specific programs. The directory *lists/* contains files that list the directories to be compiled. Directory names are relative to the $GIS directory. The file *lists/GRASS* lists all basic GRASS programs that get compiled for each machine architecture at every site. The files *lists/local* and *lists/*$ARCH, which the user can create, control site-specific and architecture-specific compilation, respectively.

$ARCH is the architecture name you approved while running the *utils/setup* script. You can determine this by running:

```
        gmake -sh | grep ARCH
```

A *lists/*$ARCH file may not exist in the distribution, but you are free to create it to add names of programs you want compiled specifically for this architecture. This architecture-specific list allows NFS-linked source code to compile a set of programs for machines which have the same architecture.

Similarly, there may not be a *lists/local* file, but you are free to create this as well. The programs in this list will compile for all machines at your local site regardless of their architecture.

The *local.example* file contains a list of display, paint, and graphics drivers provided in the source: you should uncomment the drivers you plan to use. The file *GRASS.X* contains the list of Xgrass programs.

If you want any of these items compiled, you must add them either to the *lists/local* or the *lists/*$ARCH file. You are encouraged to put the graphics drivers in the appropriate *lists/*$ARCH file.

All lists may contain comment lines, indicated by a "#" as the first character in the line.

7.5. XDRIVER

If you are compiling the XDRIVER under Openwindows on a SUN machine,[9] or if you have X11 release 3, you may have to edit the file

[9] Or on other platforms.

```
$GIS/src/display/devices/XDRIVER/XDRIVER/Gmakefile
```

and follow the instructions therein.

7.6. COMPILE GRASS

The script *GISGEN*[10] drives the compilation process. If all goes well, you will be able simply to enter the command **GISGEN** and wait. The compilation process takes from about four hours on the faster workstations to about twelve hours on the slower workstations.

It is recommended that you save *GISGEN* output in a file that you can later review for compilation warning messages. The following command runs *GISGEN*, sending the output both to the screen and to a file:

```
sh GISGEN |& tee /tmp/GISGEN.out   # csh (not sh)
```

For use in a Bourne shell, the output can be captured with

```
sh GISGEN 2> &1 | tee /tmp/GISGEN.out
```

GISGEN collects all of the directory names to be compiled from *lists/GRASS*, *lists/$ARCH*, and *lists/local* and begins running **gmake** in each directory in succession. The screen output is a collection of messages from *GISGEN* and from the UNIX **make** program. A failure at any step will halt compilation. Upon encountering a failure, you might do one of the following things:

1. Fix the compilation problem by modifying code in the directory that failed. After modification, return to this directory and re-run *GISGEN*. Compilation will resume at the failed directory and continue down the list of directories if successful.
2. Restart *GISGEN*. If the failure requires modifications to code already compiled, or the compilation options you set in step 1, you must remove *next_step/$ARCH* (or *next_step/next_step* if an architecture name was not specified when running the *setup* script). You may then re-run *GISGEN*.
3. Skip the failed directory and resume compilation with the next directory in the list. Simply run

   ```
   sh GISGEN -skip
   ```

   (You might want to capture *GISGEN* output into a file as you did earlier.)

   When complete, *GISGEN* will put the word DONE into the *next_step* file and will print the phrase "DONE generating GIS binary code" on the screen. (You may wish to review *GISGEN.out* to view any error messages.)

[10] The *utils/setup* program creates a file called *GISGEN*.< header> . We will refer to this file simply as *GISGEN*, but you will need to remember it as *GISGEN*.< header> .

7.7. LINK GRASS PROGRAMS

GISGEN directs a compilation process that places the GRASS programs in directories not directly accessible to the user community. Most user commands are actually links to a single program called *front.end.* Links to this program must be made for every actual GRASS program. This is done **after** *GISGEN* is finished. To make (or re-make) links for all user programs, run the script *MAKELINKS* (i.e., *MAKELINKS.*< header>).

7.8. COMPILE OTHER PROGRAMS

GRASS programs come in five flavors:

MAIN

The main programs are those under $GIS/*src.* These programs are compiled automatically by *GISGEN.* They have been in GRASS for at least one release, have been tested, and are reliable programs.

ALPHA

The alpha programs are those under $GIS/*src.alpha.* These programs are also compiled automatically by *GISGEN.* These programs have been compiled and given some testing but have not been part of GRASS for a full release.

CONTRIB

The contributed programs are in the directory $GIS/*src.contrib.* These programs are NOT compiled automatically by *GISGEN.* The state of these programs vary. Some may compile with **gmake**; others are more suitable as starting points for programmers who will be writing new software.

RELATED

The GRASS user community has discovered that there are several public domain programs that are very useful in conjunction with GRASS. These are found in the directory $GIS/*src.related* (assuming you have unloaded the related source code from the release media). Compile these programs based on the instructions (if available) in their respective directories.

GARDEN

The GARDEN programs are in the directory $GIS/*src.garden.* GARDEN programs are those that mix the capabilities of GRASS with the capabilities of one or more of the "related" programs (or other systems). Some require successful compilation of the "related" programs and generally compile using **gmake**. Of particular interest are the tools in *grass.informix,* which link GRASS with the Informix relational database system.

8. GRAPHICS DRIVER CONFIGURATION (etc/monitorcap)

Now you must create the file $GIS/*etc/monitorcap*, which tells the GRASS graphics application programs and graphics drivers how to communicate with one another. This file contains lines that describe each graphics driver on the system. After GRASS compilation, the $GIS/*etc/moncap.sample* file contains many different entries, all commented out (using the # symbol in the first column).[11] You must first copy the *moncap.sample* file to create a new *monitorcap* file. Then, edit the *monitorcap* file for your system's graphics devices. At a minimum, you will be uncommenting those lines that describe the graphics devices for which GRASS code has been compiled. (You identified these devices in the *lists/local* file before compiling GRASS.)

Each line in the *monitorcap* file contains six fields separated by colons:

```
name : program : description : fifos : tty : msg
```

name	The name the user uses to refer to the driver.
program	The actual program name as a UNIX command. It is specified as a relative path from $GIS (see examples).
description	A short 4- or 5-word description of the driver.
fifos	The names of the 2 fifo files that are assigned to this driver. These files must exist as named pipes,[12] have read and write permission by everybody, and not be used by any other driver (see NOTES).
tty	In some cases, it is desirable to force the driver to be started from a specific tty device. If this is the case, put the full path name of the tty in this field. Otherwise leave the field empty (see examples and NOTES following).
msg	If the tty field is specified and an attempt to start the driver is made from another tty, then this message is printed.

Examples:

[11] The original file is named *moncap.sample* to prevent subsequent compilation efforts from overwriting any existing *monitorcap* files.

[12] Named pipes are created with the command *mknod filename p*. This is done automatically for about 20 sets of fifo files by the *GISGEN* script. (The word "fifo" is short for "first-in-first-out".)

1. This first example is for a single device system, in which the monitor can be started from any terminal (tty). The two fifos are */usr/grass4.1/dev/fifo.1a* and */usr/grass4.1/dev/fifo.1b*. Here, the name of the driver is *x0*. Note that the driver program itself is specified as *driver/XDRIVER*. This is a relative path reference which will be translated into $GIS*/driver/XDRIVER*. Note that the path names of the fifos must be fully specified.

```
x0:driver/XDRIVER:Sun driver:/usr/grass4.1/dev/fifo.1a /usr/grass4.1/dev/fifo.1b::any terminal
```

2. The following example is for a two-monitor system, with the same graphics device available on each monitor. Note that the same program is used for both devices, but that the name field and the fifos field are different. Also note that both drivers must be started from a specific tty.

```
mass1:driver/MASS:Driver 1:/usr/grass4.1/dev/fifo.1a /usr/grass4.1/dev/fifo.1b:/dev/tty4:tty4
mass2:driver/MASS:Driver 2:/usr/grass4.1/dev/fifo.2a /usr/grass4.1/dev/fifo.2b:/dev/tty5:tty5
```

NOTES

1. When you are installing a driver, a pair of fifos (also known as named pipes) must exist for the driver to use. The names you choose are arbitrary. Fifo pairs with the names *fifo.1a* and *fifo.1b*, *fifo.2a* and *fifo.2b*, *fifo.3a* and *fifo.3b*, *fifo.4a* and *fifo.4b*, and *fifo.5a* and *fifo.5b* are created in the directory $GIS/*dev* by *GISGEN*.[13] It is suggested that you use these in your *monitorcap* file.
2. The tty field is not sufficiently robust to handle all situations. You should probably leave this field blank if installing GRASS on a SUN system. (In particular, SUN users running under Suntools must start the driver from the bitmap terminal, but the actual tty device number will vary from window to window.)

9. DIGITIZER DRIVER CONFIGURATION (etc/digcap)

NOTE: *The following instructions are for the new (4.1)* v.digit. *If you prefer to use the 4.0 version, see the* "APPENDIX" *at the end of this document.*

Now you must build the file $GIS/*etc/digcap*, which identifies the available digitizers and which I/O port each digitizer uses.[14]

[13] See section 7, "GRASS Compilation," for details about *GISGEN*.

[14] There are some sample *digcap* files in $GIS/*etc* that can be used as templates. To find these, use the command: **ls $GIS/etc/digcap***

Every *digcap* file should specify the "none" digitizer, which allows for on-screen digitizing using the mouse as the digitizer. In addition, there must be an entry for each digitizer locally available.

Each line in the file contains four fields separated by colons:

name : tty : driver : description

name	The name the user uses to refer to the digitizer.
tty	The tty port to be used by the digitizer.
driver	The name of the driver file (in the $GIS/*etc/digitizers* directory).
description	A short 4 or 5 word description of the digitizer.

Example:

This example is for a single digitizer system. The digitizer is an Altek digitizer on */dev/tty1*. The "none" digitizer is also specified.

```
altek:/dev/ttyb:al30f8_16: Altek digitizer, AC30, format 8
none:nodig:nofile: Run digit without the digitizer
```

NOTE: If the digitizer is later moved to a different tty port, the tty field must also be modified to reflect the change.

These next steps must be done while running as the user *root,* so **su** to *root.*

For each tty port */dev/tty*X specified in the *digcap* file, type the command:

```
chmod 0666 /dev/ttyX
```

Also be sure that no programs (e.g., **getty** or **init**) are listening on those tty ports for logins. This may require modification to the files */etc/ttys* or */etc/inittab* to disable **getty** or **init** from running on that port. See your system manager's guide for details.

Now exit from the **su** to become the user *grass* again.

10. PAINT DRIVER CONFIGURATION[15]

The GRASS program **p.map**[16] requires driver programs for each hardcopy color printer on your system that you intend to use with GRASS. With the exception of the "preview" and "preview2" drivers, which send their output to

[15] There is no $GIS/*etc/paintcap* file.

[16] and its potential replacement **p.map.new**

the graphics screen, the paint drivers send their output to tty ports named */dev/*< device> , where < device> is the driver name. For example, the "tek4695" printer uses */dev/tek4695* for its output.[17]

You will have to link these names to real device ports. For example, suppose that the "tek4695" printer is on */dev/tty10.* Link the driver to the port by typing these commands (note you may need to be logged in as *root* to create files in */dev*):

```
ln /dev/tty10 /dev/tek4695
chmod 0666 /dev/tek4695
```

Also be sure that no programs (e.g., **getty** or **init**) are listening on those tty ports for logins. This may require modification to the files */etc/ttys* or */etc/inittab* to disable **getty** or **init** from running on that port. See your system manager's guide for details.

The paint driver configuration design supports multiple printers of different types. For example, it is permissible to have both a "tek4695" and a "shinko635" printer on the system.

Multiple printers of the same type may be connected to a single system. Similarly, remote printers can also be supported. You must familiarize yourself with the paint support files and directories to enable such capabilities. The support files and directories, kept in the directory $GIS*/etc/paint*, are:

driver.sh

A directory containing shell scripts that set driver variables and then generally call identically named programs in the *driver* directory.

driver

A directory containing the actual binary programs that process the data and generate output to the devices.

driver.rsh

A shell script that can be used in situations where the paint device is supported by GRASS on a remote machine.

When a user selects a driver, the selection is made from names in the *driver.sh* directory. To provide for a second or third printer of a common type, simply make a new entry in this directory. For example, say there is a *tek4695* file already here, but you want to support two "tek4695" printers. Copy the *tek4695* shell script to a new file, say, *tek4695b.* You now must edit this new file and identify the tty device of the second printer. Let us assume that you decided to run this device from

[17] The "tek4695" and "shinko635" printers perform parallel i/o only. If your machine does not have a parallel port, you will have to use a serial to parallel converter.

/dev/tek4695b. Edit the new script, changing only the last line from:

```
exec ${PAINT_DRIVER?}
```

to

```
exec ${GISBASE?}/etc/paint/driver/tek4695
```

The rest of the script should remain untouched. For further details on the operation of the paint driver, refer to the *GRASS 4.1 Programmer's Manual.*

If a machine running GRASS needs access to a printer on a remote machine that is available via a local network, you can make other modifications to the *driver.sh* files. For example, imagine the following situation:

machine A	Runs GRASS 4.1
	has tek4695 printer connected to */dev/tek4695*
machine B	Runs GRASS 4.1
	needs access to machine A's tek4695 printer

On machines A and B, edit the *driver.sh/tek4695* files (note that the files may be shared across an NFS mount if the machines are the same architecture, e.g., two SUN-4 machines). Change the following lines:

```
# for networked machines, set the host which has the printer
# printer_host= . print_host_alias= .
# case `hostname` in
#       $printer_host|$printer_host_alias);;
#       *) exec rsh $printer_host $GISBASE/etc/paint/driver.rsh $PAINTER n
#          exit 0;;
# esac
```

Uncomment these lines and add machine names where appropriate:

```
# for networked machines, set the host which has the printer
printer_host= A print_host_alias= A.*
case `hostname` in
     $printer_host$printer_host_alias);;
     *) exec rsh $printer_host $GISBASE/etc/paint/driver.rsh $PAINTER n
        exit 0::
esac
```

If $GISBASE is not identical on the two machines, change the reference to $GISBASE in the above lines to the actual full path name of $GISBASE on machine A.

This script, when executed on machine A, skips the edited session and behaves normally. When run on machine B (or any machine other than A), a UNIX **rsh** connection is made to machine A, where the printer will be operated. Note that the **rsh** command requires that the individual users have logins on both machines and have the ability (through the *.rhosts* file in their home directories) to use **rlogin** and **rsh** between the two machines without passwords.

NOTE: The command **hostname** is used in this example. If you machine does not have a **hostname** command, you will have to substitute some script or program that returns the name of the machine (e.g., **uname -n**).

11. SUPPLY YOUR GRASS 4.0 DATA

Do you have GRASS 4.0 data you wish to provide to GRASS 4.1? If so, we suggest that you **copy** your 4.0 data into the 4.1 data directory (which you have already defined as $GISDBASE). Use the following instructions as a guide; you will need to modify them to fit the actual location of files on your system.

Note that your data is probably the most expensive component of your GRASS system – more expensive than the hardware, the training, the system support, and the extra air conditioning. It is our policy at CERL always to have at least two copies of data. This can be two copies on disk, two on tape, or on a combination of tape and disk. Keeping three or more copies, some stored in different buildings, provides necessary backups.

Using ½" tape

1. Login as *root*. (This must be done as *root*, otherwise the ownership of the database files will change.)

2. Change directory to your GRASS 4.0 data directory. For example,

```
cd /usr/grass/data
```

3. Use **tar** to copy the contents to tape. Note that this directory will contain the names of all the locations your data covers. You will **not** want to back up the **spearfish** database. Assuming your data locations are "spearfish." "county." "park," and "base," and your tape drive is */dev/rmt0,* you will use the following command:

```
tar cf /dev/rmt0 county park base
```

4. If you do not have enough disk space for two copies of your data, you will want to remove the 4.0 data at this point:

```
rm -rf county park base
```

However. if you do remove the data. you should first rerun step 3 with a second tape. just in case the first tape is written badly or becomes damaged.

5. Change directory to the GRASS 4.1 data directory:

```
cd $GISDBASE
```

(You set $GISDBASE early in the installation process. You may have to do this again if you have logged out and logged in again since the installation.)

6. Copy the data from tape back to the new directory. If you loaded the GRASS 4.1 **spearfish** database earlier in the installation process, you should find the directory *spearfish* in this directory. If all is well, read the data from tape:

```
mt -t /dev/rmt0 rew  (this command is probably not necessary)
tar xvpf /dev/rmt0
```

(In this example, replace "/dev/rmt0" with the device name of your tape drive.)

Copying direct disk to disk:

You may want to skip the tape step. If you have enough disk space to hold two copies of the data simultaneously, you can do the following:

1. Login as *root*. (This must be done as *root*, otherwise the ownership of the database files will change.)

2. Copy the data from the old directory to the new one. For example, assume the old data directory is */usr/grass/data* and the new is $GISDBASE, and assume the same databases as in the above example. Issue the following command:

```
cd /usr/grass/data
tar cf - county park base |(cd $GISDBASE; tar xvpf -)
```

12. DID YOU MISS SOMETHING?

In the process of installation you may have missed something. Some of the more common steps skipped or missed can be completed without going through the entire installation. The more common steps are mentioned here.

Bad compilation

The results of the compilation may not work for a variety of reasons. Past experience has shown that bad hardware, new compilers, different floating point processors, different hardware, and wrong information in the GRASS files in the $GIS/*src/CMD* directory can cause problems. If the problem is suspected to be wrong information, re-examine the "System Configuration" section.

Incomplete compilation

The $GIS/*src/CMD/lists* directory contains the file *GRASS* and, if you created them, the files *local* and *$ARCH*. These contain names of the directories that *GISGEN* compiles. It is possible that you:

1. told *GISGEN* to skip one of the directories; or
2. did not add a graphics driver to *src/CMD/lists/local*

You can compile the code in any directory with the following steps:

1. **cd** to that directory
2. run **gmake**

Sample data missing

The data on the release tape can be loaded at any time. Use the instructions in the "Extracting the Source Code and Sample Database," section 7, as a guide.

13. GRASS INSTALLATION IS COMPLETE

At this point, GRASS is now installed on the system and can be run using the command **grass4.1.** This command will be found in the directory you specified during *setup.* You will find this directory name defined in the < header> file as UNIX_BIN.

14. CONTRIB-TEST AND OTHER PROGRAMS

In addition to the supported code under $GIS/*src* and $GIS/*src.alpha,* there are unsupported programs under $GIS.

Directory	Contents
src.contrib	GRASS programs useful to programmers
src.related	Programs that can work with GRASS
src.garden	Programs that mix GRASS and other systems' capabilities

These programs are not automatically compiled because they are not supported by OGI. (OGI's intention is to make these unsupported programs available. Compile and use them **at your own risk.**) If there are instructions within these directories, they should assist you in making decisions about what and how to compile.

15. CLEANING UP

If you are short of disk space you may want to remove some of the non-essential parts of GRASS.

Object files

Unless you will be actively changing all of the code, you will save at least six megabytes by removing all object (.o) files. This can be done with the command:

```
find $GIS/src* -name '*.o' -print |xargs rm -f
```

or (if your system does not have **xargs**),

```
find SGIS/src* -name '*.[o]' -exec rm -f {} \;
```

Source files

You can save about thirty megabytes of additional space by simply removing the source and manual directories:

```
rm  -rf  SGIS/src* SGIS/man
```

This will in no way affect the operation of GRASS. Note, however, that you should **not** remove SGIS*/man* if you have set up $GISBASE and $SRC to be in the same directory tree (since, in this case, deleting $GIS*/man* will delete the help files used by GRASS).

16. MOVING THE BINARY INSTALLATION

If you need to move GISBASE to another directory, it is no longer necessary to recompile the system. Common reasons for moving the programs include disk reorganization and installation on other systems.

For example, assume you wish to move $GISBASE from */usr/grass4.1* to */home/grass4.1*. To accomplish this, do the following steps.

1. Move the base directory. On most systems, *root* can accomplish this with the command:

   ```
   mv  /usr/grass4.1  /home/grass4.1
   ```

   On others a copy will be necessary:

   ```
   (cd /usr; tar cpf - grass4.1) |(cd /home; tar xpf -)
   ```

   This command, if done as *root*, will preserve the original file ownerships and permissions.

2. Edit *grass4.1* (this for the user). Replace all references to the old directory with references to the new directory. For example, this script might be changed from:

   ```
   :
   GISBASE= /usr/grass4.1
   export GISBASE
   exec SGISBASE/etc/GIS.sh
   ```

   to:

   ```
   :
   GISBASE= /home/grass4.1
   export GISBASE
   exec SGISBASE/etc/GIS.sh
   ```

3. Edit the *monitorcap* file. In this example, the new GISBASE is */home/grass4.1*, so you would edit SGISBASE/*etc/monitorcap* and change any references to */usr* to refer to */home*.

17. APPENDIX

This appendix contains miscellaneous information about various other platforms as well as instructions for installing and using the 4.0 version of **v.digit**.

17.1. MISSING UNIX COMMANDS

GRASS is written assuming the existence of certain UNIX commands which may not be present on your system. These commands are listed in the table below. If you system does not have these commands, you will find shell scripts under $GIS/CMD/utils which emulate the functionality required. You can copy these scripts to UNIX_BIN (or any other directory that will be in everybody's PATH variable) giving them the correct name:

command	function	replacement[19]
clear	clears the terminal screen	clear.tput
tset	terminal setup	tset.tput
reset	terminal reset	reset.tput
more	pages displayed text	more.pg
whoami	prints user's name	whoami.sh
lpr	print command	< none supplied>

17.2. XDRIVER REQUIREMENTS

The XDRIVER as developed by CERL requires that the default visual be a Pseudo-color visual with at least 256 colors. If your X server does not set the default visual to this type, then you must teach it to do so. It is beyond the scope of this document to provide such instructions about your X server. However, we do know that for the Data General AViiON you can do this by editing the file */var/X11/xdm/Xservers* and placing the following line into this file:

```
:0 local /usr/bin/X11/X :0 bc -cc 3
```

17.3. XGRASS/OPENWINDOWS REQUIREMENTS

To make XGRASS function properly in a SUN Openwindows environment, you must make sure that the Motif keysyms are installed. To do this, check to see if the file */usr/openwin/lib/XKeysymDB* exists. If it does, append the file *src/xgrass/XKeysymDB* to it; if it does not, copy the file *src/xgrass/XKeysymDB* to

[19] The scripts clear.tput. tset.tput and reset.tput require the UNIX tput(1) command: the script more.pg requires the UNIX pg(1) command: and the whoami.sh command requires the Bourne Shell and the /tmp directory. There is no replacement for lpr because the command to print ascii text to a printer varies. You will have to write this one.

the directory */usr/openwin/lib.* This may require root permission. It is not clear how to fix the problem if you cannot do this. The error comes from Xt translation table parsing and cannot be fixed in Motif; this problem arises because the Motif keysyms have not been installed in the Openwindows server.

17.4. SCO UNIX

There may be problems compiling *src/libes/gis* under SCO UNIX. The **make** program on this system does not seem to be able to handle the length of the command which builds the library. The workaround is to execute the command by hand (after *GISGEN* fails), and then rerun *GISGEN*. To execute the command directly:

```
cd $GIS/src/libes/gis
ar rc ../LIB.$ARCH/libgis.a OBJ.$ARCH/*.o
```

where $ARCH is the architecture name you defined during *setup.*

17.5. CRAY LOADER PROBLEMS

The Cray UNICOS loader does not know that an archive library is a true library unless the library name is preceded by "-l". Since GRASS programs are compiled with the full name of the library (rather than using the "-l" flag) this will cause the loader to either not load the library or to load all the object files in the library. To get around this you will need to write your own **cc** command which prepends the "-l" flag to arguments that end in ".a" and then runs the real C compiler with the modified arguments.

17.6. INTERGRAPH HARDWARE PLATFORMS

There are two new programs that will not compile on the Intergraph hardware platform due to portability issues. Those users compiling on an Intergraph should be aware that the compilation process will be stopped at these two programs, or they should delete the program names from the list being used by the *GISGEN* process before beginning compilation. These programs are:

```
src.alpha/imagery/i.ortho.photo
src.alpha/mapdev/v.in.tig.lndmk
```

17.7. MIPS/CONTROL DATA HARDWARE PLATFORMS (RiscOS, EP/IX)

When running *setup*, be sure to specify */usr/bsd43/bin/cc* as the compiler to be used and specify the full path name.

After running *setup*, edit the *head/*$ARCH file to change the lines

```
        XLIBPATH        = -L/usr/lib
        XTLIBPATH   = -L/usr/lib
        XMLIBPATH   = -L/usr/lib
```

to read

```
        XLIBPATH        =
        XTLIBPATH   =
        XMLIBPATH   =
```

(i.e., remove the "-L/usr/lib" from each of these lines).

The Mips/CDC BSD compilation environment does not provide certain C header files that are required by some GRASS programs; these must be made available by taking the following steps after running *setup.* (Note that you should substitute "mips" in these examples with the values of $ARCH, i.e., the name you gave to this architecture in *setup*).

```
        mkdir $SRC/include/mips
        cp /usr/include/posix/stdlib.h $SRC/include/mips
        cp /usr/include/posix/unistd.h $SRC/include/mips
```

Add "-I$(SRC)/include/$(ARCH)" to the COMPILE_FLAGS variable defined in *head/*$ARCH (or specify it during *setup*). Finally, edit $SRC/*include/mips/stdlib.h* to add the line

```
        extern double strtod();
```

as the second line before the end of the file (i.e., this should not be the last line of the file!).

The Mips/CDC C compiler does not understand the "const" keyword, which is used in some GRASS programs, and using it causes a fatal compilation error. To work around this problem, add "-Dconst= " to the COMPILE_FLAGS variable defined in *head/*$ARCH (or specify it during *setup*).

The Mips/CDC BSD compilation environment does not provide certain standard UNIX library routines that are required to build some GRASS programs. These muse be made available by adding them to the "gis" library. Follow the regular compilation steps by running *GISGEN.* After the "gis" library has been built, interrupt the compilation process and take the following steps:

```
        cd $SRC/libes/gis/LIB.$ARCH
        ar x /usr/lib/libc.a strtod.o ctype.o
        ar ruv libgis.a strtod.o ctype.o
        cd $SRC/src/CMD
```

Note also that some XGRASS programs may need to have the order of the libraries specified in their *Gmakefiles* changed in order to find these functions (i.e. you may need to re-order the library list to place "$(GISLIB)" at the end of the list.

17.8. SAMPLE HEADER FILES

Following are sample < header> files created during the preparation of this release.

```
SUN4
CC                   = cc
ARCH                 = sun4
GISBASE              = /grass4.1b/sun4
UNIX_BIN             = /usr/local/bin
DEFAULT_DATABASE     = /grass.data
DEFAULT_LOCATION     = spearfish
COMPILE_FLAGS        = -O
LDFLAGS              = -s
XCFLAGS              = -D_NO_PROTO
XLDFLAGS             =
XINCPATH             =
XMINCPATH            =
XLIBPATH             = -L/usr/lib
XTLIBPATH            = -L/usr/lib
XMLIBPATH            = -L/usr/lib
XEXTRALIBS           =
TERMLIB              = -ltermlib
CURSES               = -lcurses $(TERMLIB)
MATHLIB              = -lm
LIBRULE              = ar ruv $@ $?; ranlib $@
USE_TERMIO           =
USE_MTIO             = -DUSE_MTIO
USE_FTIME            = -DUSE_FTIME
DIGITFLAGS           = -DUSE_SETREUID -DUSE_SETEUID -DUSE_SETPRIORITY
VECTLIBFLAGS         = -DPORTABLE_3
GETHOSTNAME          = -DGETHOSTNAME_OK
```

Intergraph Interpro		
CC	=	acc
ARCH	=	ig
GISBASE	=	/grass4.1b/ig
UNIX_BIN	=	/usr/local/bin
DEFAULT_DATABASE	=	/grass.data
DEFAULT_LOCATION	=	spearfish
COMPILE_FLAGS	=	-O -w -knr
LDFLAGS	=	-s
XCFLAGS	=	-DIGRAPH -DSYSV -D_NO_PROTO
XLDFLAGS	=	
XINCPATH	=	
XMINCPATH	=	
XLIBPATH	=	
XTLIBPATH	=	-L/usr/lib
XMLIBPATH	=	-L/usr/lib
XEXTRALIBS	=	-lbsd -lc -s
TERMLIB	=	-ltermlib
CURSES	=	-lcurses $(TERMLIB)
MATHLIB	=	-lm
LIBRULE	=	ar ruv $@ $?
USE_TERMIO	=	-DUSE_TERMIO
USE_MTIO	=	-DUSE_MTIO
USE_FTIME	=	
DIGITFLAGS	=	-DINTERPRO
VECTLIBFLAGS	=	
GETHOSTNAME	=	-DGETHOSTNAME_UNAME

Data General AViiON		
CC	=	cc
ARCH	=	aviion
GISBASE	=	/grass4.1b/aviion
UNIX_BIN	=	/usr/local/bin
DEFAULT_DATABASE	=	/grass.data
DEFAULT_LOCATION	=	spearfish
COMPILE_FLAGS	=	-O
LDFLAGS	=	-s
XCFLAGS	=	-D_NO_PROTO
XLDFLAGS	=	
XINCPATH	=	
XMINCPATH	=	
XLIBPATH	=	-L/usr/lib
XTLIBPATH	=	-L/usr/lib
XMLIBPATH	=	-L/usr/lib
XEXTRALIBS	=	-lPW
TERMLIB	=	
CURSES	=	-lcurses $(TERMLIB)
MATHLIB	=	-lm
LIBRULE	=	ar ruv $@ $?
USE_TERMIO	=	-DUSE_TERMIO
USE_MTIO	=	-DUSE_MTIO
USE_FTIME	=	-DUSE_FTIME
DIGITFLAGS	=	-DUSE_SETREUID -DUSE_SETEUID -DUSE_SETPRIORITY
VECTLIBFLAGS	=	
GETHOSTNAME	=	-DGETHOSTNAME_OK

Silicon Graphics IRIS		
CC	=	cc
ARCH	=	sgi
GISBASE	=	/GRASS.bin/4.1
UNIX_BIN	=	/usr/local/bin
DEFAULT_DATABASE	=	/net/GRASS.src/4.1/data
DEFAULT_LOCATION	=	spearfish
COMPILE_FLAGS	=	-cckr -O
LDFLAGS	=	-s
XCFLAGS	=	-D_NO_PROTO
XLDFLAGS	=	
XINCPATH	=	
XMINCPATH	=	
XLIBPATH	=	-L/usr/lib
XTLIBPATH	=	-L/usr/lib
XMLIBPATH	=	-L/usr/lib
XEXTRALIBS	=	-lPW
TERMLIB	=	-ltermlib
CURSES	=	-lcurses $(TERMLIB)
MATHLIB	=	-lm
LIBRULE	=	ar ruv $@ $?
USE_TERMIO	=	-DUSE_TERMIO
USE_MTIO	=	-DUSE_MTIO
USE_FTIME	=	
DIGITFLAGS	=	-DUSE_SETREUID -DUSE_SETEUID -DUSE_SETPRIORITY
VECTLIBFLAGS	=	
GETHOSTNAME	=	-DGETHOSTNAME_OK

Mips RISCos or CDC EP/IX

```
CC                  = /usr/bsd43/bin/cc
ARCH                = mips
GISBASE             = /GRASS.bin/4.1final/mips
UNIX_BIN            = /usr/local/bin
DEFAULT_DATABASE    = /GRASS.src/4.1final/data
DEFAULT_LOCATION    = spearfish
COMPILE_FLAGS       = -O -traditional -Dconst=  -I$(SRC)/include/$(ARCH)
LDFLAGS             = -s
XCFLAGS             = -Olimit 2000 -Wf,-XNd8400,-XNp12000 -D_NO_PROTO
XLDFLAGS            =
XINCPATH            =
XMINCPATH           =
XLIBPATH            =
XTLIBPATH           =
XMLIBPATH           =
XLIB                = -lX11
XTLIB               = -lXt
XMLIB               = -lXm
XEXTRALIBS          =
TERMLIB             = -ltermlib
CURSES              = -lcurses $(TERMLIB)
MATHLIB             = -lm
LIBRULE             = ar ruv $@ $?
USE_TERMIO          =
USE_MTIO            = -DUSE_MTIO
USE_FTIME           = -DUSE_FTIME
DIGITFLAGS          = -DUSE_SETREUID -DUSE_SETEUID -DUSE_SETPRIORITY
VECTLIBFLAGS        =
GETHOSTNAME         = -DGETHOSTNAME_OK
```

17.9. GRASS 4.0 DIGITIZER DRIVER CONFIGURATION--etc/digitcap

If you prefer to use the 4.0 digitizer driver configuration, use the following instructions instead *of those provided in Section 9.*

You will need to make sure that the following directories are compiled by *GISGEN*

```
src/mapdev/bin_dig
src/mapdev/v.digit2
src/mapdev/digitizers/none
```

as well as any 4.0 digitizer driver(s) for the digitizer(s) you have.

Now you must build the file $GIS/*etc/digitcap,* which identifies the available digitizers and the i/o port each digitizer uses.[20]

Every *digitcap* file should specify the "none" digitizer, which allows for on-screen digitizing using the mouse as the digitizer. In addition, there must be an entry for each digitizer locally available.

Each line in the file contains four fields separated by colons:

name : tty : program : description

name	The name the user uses to refer to the digitizer
tty	The tty port to be used by the digitizer
program	The actual program name as a UNIX command. It is specified as *digit.*
description	A short 4- or 5-word description of the digitizer.

Example:

This example is for a single digitizer system. The digitizer is a "Kurta" digitizer on */dev/tty1.* The "none" digitizer is also specified.

```
kurta:/dev/tty1:KURTA digitizer 9600 baud
none:nodig:digit: Run digit without the digitizer
```

NOTE: If the digitizer is later moved to a different tty port, the tty field must also be modified to reflect the change.

These next steps must be done while running as the user *root,* so **su** to *root.*

For each tty device */dev/tty*X specified in the *digitcap* file, type:

```
chmod 0666 /dev/ttyX
```

Also be sure that no programs (e.g., **getty** or **init**) are listening on those tty ports for logins. This may require modification to the files */etc/ttys* or */etc/inittab*) to disable **getty** or **init** from running on that port. See your system manager's guide for details.

Now exit from **su** to become the user *grass* again.

[20] There are some sample *digitcap* files in $GIS/*etc* that can be used as templates. To find these: ls $GIS/etc/digitcap* .

APPENDIX B

TCL 7.3/TK 3.6 (expect 5.7) INSTALLATION MANUAL

Tcl

by John Ousterhout
University of California at Berkeley
ouster@cs.berkeley.edu

1. Introduction

This directory contains the sources and documentation for Tcl, an embeddable tool command language. The information here corresponds to release 7.0.

2. Documentation

The best way to get started with Tcl is to read the draft of my upcoming book on Tcl and Tk, which can be retrieved using anonymous FTP from the directory "tcl" on sprite.berkeley.edu. Part I of the book provides an introduction to writing Tcl scripts and Part III describes how to write C code that uses the Tcl C library procedures.

The "doc" subdirectory in this release contains a complete set of manual entries for Tcl. Files with extension ".1" are for programs (for example, tclsh.1); files with extension ".3" are for C library procedures; and files with extension ".n" describe Tcl commands. The file "doc/Tcl.n" gives a quick summary of the Tcl language syntax. To print any of the man pages, cd to the "doc" directory and invoke your favorite variant of troff using the normal -man macros, for example

```
ditroff -man Tcl.n
```

to print Tcl.n. If Tcl has been installed correctly and your "man" program supports it, you should be able to access the Tcl manual entries using the normal "man" mechanisms, such as

```
man Tcl
```

3. Compiling and installing Tcl

This release should compile and run "out of the box" on any UNIX-like system that approximates POSIX, BSD, or System V. I know that it runs on workstations from Sun, DEC, H-P, IBM, and Silicon Graphics, and on PC's running SCO UNIX and Xenix. To compile Tcl, do the following:

(a) Type "./configure" in this directory. This runs a configuration script created by GNU autoconf, which configures Tcl for your system and creates a Makefile. The configure script allows you to customize the Tcl configuration for your site; for details on how you can do this, see the file "configure.info".

(b) Type "make". This will create a library archive called "libtcl.a" and an interpreter application called "tclsh" that allows you to type Tcl commands interactively or execute script files.

(c) If the make fails then you'll have to personalize the Makefile for your site or possibly modify the distribution in other ways. First check the file "porting.notes" to see if there are hints for compiling on your system. If you need to modify Makefile, there are comments at the beginning of it that describe the things you might want to change and how to change them.

(d) Type "make install" to install Tcl binaries and script files in standard places. You'll need write permission on /usr/local to do this. See the Makefile for details on where things get installed.

(e) At this point you can play with Tcl by invoking the "tclsh" program and typing Tcl commands. However, if you haven't installed Tcl then you'll first need to set your TCL_LIBRARY variable to hold the full path name of the "library" subdirectory.

If you have trouble compiling Tcl, I'd suggest looking at the file "porting.notes". It contains information that people have sent me about changes they had to make to compile Tcl in various environments. I make no guarantees that this information is accurate, complete, or up-to-date, but you may find it useful. If you get Tcl running on a new configuration, I'd be happy to receive new information to add to "porting.notes". I'm also interested in hearing how to change the configuration setup so that Tcl compiles on additional platforms "out of the box".

4. Test suite

There is a relatively complete test suite for all of the Tcl core in the subdirectory "tests". To use it just type "make test" in this directory. You should then see a printout of the test files processed. If any errors occur, you'll see a much more substantial printout for each error. See the README file in the "tests" directory for more information on the test suite.

5. Summary of changes in this release

Tcl 7.0 is a major new release that includes several new features and a few incompatible changes. For a complete list of all changes to Tcl in chronological order, see the file "changes". Those changes likely to cause compatibility problems with existing C code or Tcl scripts are specially marked. The most important changes are summarized below.

Tcl configuration and installation has improved in several ways:

1. GNU autoconf is now used for configuring Tcl prior to compilation.

2. The "tclTest" program no longer exists. It has been replaced by "tclsh", which is a true shell-like program based around Tcl (tclTest didn't really work very well as a shell). There's a new program "tcltest" which is the same as "tclsh" except that it includes a few extra Tcl commands for testing purposes.

3. A new procedure Tcl_AppInit has been added to separate all of the application-specific initialization from the Tcl main program. This should make it easier to build new Tcl applications that include extra packages.

4. There are now separate manual entries for each of the built-in commands. The manual entry "Tcl.n", which used to describe all of the built-ins plus many other things, now contains a terse but complete description of the Tcl language syntax.

Here is a list of all incompatibilities that affect Tcl scripts:

1. There have been several changes to backslash processing:
   - Unknown backslash sequences such as "*" are now replaced with the following character (such as "*"); Tcl used to treat the backslash as an ordinary character in these cases, so both the backslash and the following character would be passed through.
   - Backslash-newline now eats up any white space after the newline, replacing the whole sequence with a single space character. Tcl used to just remove the backslash and newline.

- The obsolete sequences \Cx, \Mx, \CMx, and \e no longer get special treatment.
- The "format" command no longer does backslash processing on its input string.

You can invoke the shell command below to locate backslash uses that may potentially behave differently under Tcl 7.0. This command will print all of the lines from the script files "*.tcl" that may not work correctly under Tcl 7.0:

```
    egrep '(\\$)|(\\[^][bfnrtv\0-9{}$ ;"])' *.tcl
```

In some cases the command may print lines that are actually OK.

2. The "glob" command now returns only the names of files that actually exist, and it only returns names ending in "/" for directories.

3. When Tcl prints floating-point numbers (e.g. in the "expr" command) it ensures that the numbers contain a "." or "e" so that they don't look like integers.

4. The "regsub" command now overwrites its result variable in all cases. If there is no match, then the source string is copied to the result.

5. The "exec", "glob", "regexp", and "regsub" commands now include a "--" switch; if the first non-switch argument starts with a "-" then there must be a "--" switch or the non-switch argument will be treated as a switch.

6. The keyword "UNIX" in the variable "errorCode" has been changed to "POSIX".

7. The "format" and "scan" commands no longer support capitalized conversion specifiers such as "%D" that aren't supported by ANSI sprintf and sscanf.

Here is a list of all of the incompatibilities that affect C code that uses the Tcl library procedures. If you use an ANSI C compiler then any potential problems will be detected when you compile your code: if your code compiles cleanly then you don't need to worry about anything.

1. Tcl_TildeString now takes a dynamic string as an argument, which is used to hold the result.

2. tclHash.h has been eliminated; its contents are now in tcl.h.

3. The Tcl_History command has been eliminated: the "history" command is now automatically part of the interpreter.

4. The Tcl_Fork and Tcl_WaitPids procedures have been deleted (just use fork and waitpid instead).

5. The "flags" and "termPtr" arguments to Tcl_Eval have been eliminated, as has the "noSep" argument to Tcl_AppendElement and the TCL_NO_SPACE flag for Tcl_SetVar and Tcl_SetVar2.

6. The Tcl_CmdBuf structure has been eliminated, along with the procedures Tcl_CreateCmdBuf, Tcl_DeleteCmdBuf, and Tcl_AssembleCmd. Use dynamic strings instead.

7. Tcl_SetVar and Tcl_UnsetVar2 now return TCL_OK or TCL_ERROR instead of 0 or -1.

8. Tcl_UnixError has been renamed to Tcl_PosixError.

9. Tcl no longer redefines the library procedures "setenv", "putenv", and "unsetenv" by default. You have to set up special configuration in the Makefile if you want this.

Below is a sampler of the most important new features in Tcl 7.0. Refer to the "changes" file for a complete list.

1. The "expr" command supports transcendental and other math functions, plus it allows you to type expressions in multiple arguments. Its numerics have also been improved in several ways (e.g. support for NaN).

2. The "format" command now supports XPG3 %n$ conversion specifiers.

3. The "exec" command supports many new kinds of redirection such as >> and >&, plus it allows you to leave out the space between operators like < and the file name. For processes put into the background, "exec" returns a list of process ids.

4. The "lsearch" command now supports regular expressions and exact matching.

5. The "lsort" command has several new switches to control the sorting process (e.g. numerical sort, user-provided sort function, reverse sort, etc.).

6. There's a new command "pid" that can be used to return the current process ids or the process ids from an open file that refers to a pipeline.

7. There's a new command "switch" that should now be used instead of "case". It supports regular expressions and exact matches, and also uses single patterns instead of pattern lists. "Case" is now deprecated, although it's been retained for compatibility.

8. A new dynamic string library has been added to make it easier to build up strings and lists of arbitrary length. See the manual entry "DString.3".

9. Variable handling has been improved in several ways: you can now use whole-array traces to create variables on demand, you can delete variables during traces, you can upvar to array elements, and you can retarget an upvar variable to stop through a sequence of variables. Also, there's a new library procedure Tcl_LinkVar that can be used to associate a C variable with a Tcl variable and keep them in sync.

10. New library procedures Tcl_SetCommandInfo and Tcl_GetCommandInfo allow you to set and get the clientData and callback procedure for a command.

11. Added "-errorinfo" and "-errorcode" options to "return" command; they allow much better error handling.

12. Made prompts in tclsh user-settable via "tcl_prompt1" and "tcl_prompt2" variables.

13. Added low-level support that is needed to handle signals: see Tcl_AsyncCreate, etc.

6. Tcl newsgroup

There is a network news group "comp.lang.tcl" intended for the exchange of information about Tcl, Tk, and related applications. Feel free to use the newsgroup both for general information questions and for bug reports. I read the newsgroup and will attempt to fix bugs and problems reported to it.

7. Tcl contributed archive

Many people have created exciting packages and applications based on Tcl and made them freely available to the Tcl community. An archive of these contributions is kept on the machine harbor.ecn.purdue.edu. You can access the archive using anonymous FTP; the Tcl contributed archive is in the directory "pub/tcl". The archive also contains an FAQ ("frequently asked questions") document that provides solutions to problems that are commonly encountered by TCL newcomers.

8. Support and bug fixes

I'm very interested in receiving bug reports and suggestions for improvements. Bugs usually get fixed quickly (particularly if they are serious), but enhancements may take a while and may not happen at all unless there is widespread support for them (I'm trying to slow the rate at which Tcl turns into a kitchen sink). It's almost impossible to make incompatible changes to Tcl at this point.

The Tcl community is too large for me to provide much individual support for users. If you need help I suggest that you post questions to comp.lang.tcl. I read the newsgroup and will attempt to answer esoteric questions for which no-one else is likely to know the answer. In addition, Tcl support and training are available commercially from NeoSoft. For more information, send e-mail to "info@neosoft.com".

9. Tcl release organization

Each Tcl release is identified by two numbers separated by a dot, e.g. 6.7 or 7.0. If a new release contains changes that are likely to break existing C code or Tcl scripts then the major release number increments and the minor number resets to zero: 6.0, 7.0, etc. If a new release contains only bug fixes and compatible changes, then the minor number increments without changing the major number, e.g. 7.1, 7.2, etc. If you have C code or Tcl scripts that work with release X.Y, then they should also work with any release X.Z as long as Z > Y.

Beta releases have an additional suffix of the form bx. For example, Tcl 7.0b1 is the first beta release of Tcl version 7.0, Tcl 7.0b2 is the second beta release, and so on. A beta release is an initial version of a new release, used to fix bugs and bad features before declaring the release stable. Each new release will be preceded by one or more beta releases. I hope that lots of people will try out the beta releases and report problems back to me. I'll make new beta releases to fix the problems, until eventually there is a beta release that appears to be stable. Once this occurs I'll remove the beta suffix so that the last beta release becomes the official release.

If a new release contains incompatibilities (e.g. 7.0) then I can't promise to maintain compatibility among its beta releases. For example, release 7.0b2 may not be backward compatible with 7.0b1. I'll try to minimize incompatibilities between beta releases, but if a major problem turns up then I'll fix it even if it introduces an incompatibility. Once the official release is made then there won't be any more incompatibilities until the next release with a new major version number.

10. Compiling on non-UNIX systems

The Tcl features that depend on system calls peculiar to UNIX (stat, fork, exec, times, etc.) are now separate from the main body of Tcl, which only requires a few generic library procedures such as malloc and strcpy. Thus it should be relatively easy to compile Tcl for non-UNIX machines such as MACs and DOS PC's, although a number of UNIX-specific commands will be absent (e.g. exec, time, and glob). See the comments at the top of Makefile for information on how to compile without the UNIX features.

The Tk Toolkit

by John Ousterhout
University of California at Berkeley
ouster@cs.berkeley.edu

1. Introduction

This directory contains the sources and documentation for Tk, an X11 toolkit that provides the Motif look and feel and is implemented using the Tcl scripting language. The information here corresponds to Tk 3.6. It is designed to work with Tcl 7.3 and may not work with other releases of Tcl.

2. Documentation

The best way to get started with Tk is to read the draft of my upcoming book on Tcl and Tk, which can be retrieved using anonymous FTP from the directory "ucb/tcl" on ftp.cs.berkeley.edu. Part II of the book provides an introduction to writing Tcl scripts for Tk and Part IV describes how to build new widgets and geometry managers in C using Tk's library procedures.

The "doc" subdirectory in this release contains a complete set of manual entries for Tk. Files with extension ".1" are for programs such as wish; files with extension ".3" are for C library procedures; and files with extension ".n" describe Tcl commands. To print any of the manual entries, cd to the "doc" directory and invoke your favorite variant of troff using the normal -man macros, for example

```
ditroff -man wish.1
```

to print wish.1. If Tk has been installed correctly and your "man" program supports it, you should be able to access the Tcl manual entries using the normal "man" mechanisms, such as

```
man wish
```

3. Compiling and installing Tk

This release should compile and run with little or no effort on any UNIX-like system that approximates POSIX, BSD, or System V and runs the X Window System. I know that it runs on workstations from Sun, DEC, H-P, IBM, and Silicon Graphics, and on PC's running SCO UNIX and Xenix. To compile Tk, do the following:

(a) Make sure that this directory and the corresponding release of Tcl are both subdirectories of the same directory. This directory should be named tk3.6 and the Tcl release directory should be named tcl7.3.

(b) Type "./configure" in this directory. This runs a configuration script created by GNU autoconf, which configures Tcl for your system and creates a Makefile. The configure script allows you to customize the Tk configuration for your site; for details on how you can do this, see the file "configure.info".

(c) Type "make". This will create a library archive called "libtk.a" and an interpreter application called "wish" that allows you to type Tcl commands interactively or execute script files.

(d) If the make fails then you'll have to personalize the Makefile for your site or possibly modify the distribution in other ways.

First check the file "porting.notes" to see if there are hints
    for compiling on your system. If you need to modify Makefile,
    there are comments at the beginning of it that describe the things
    you might want to change and how to change them.

(e) Type "make install" to install Tk's binaries and script files in
    standard places. In the default configuration information will
    be installed in /usr/local so you'll need write permission on
    this directory.

(f) At this point you can play with Tcl by invoking the "wish"
    program and typing Tcl commands. However, if you haven't installed
    Tk then you'll first need to set your TK_LIBRARY environment
    variable to hold the full path name of the "library" subdirectory.
    If you haven't installed Tcl either then you'll need to set your
    TCL_LIBRARY environment variable as well (see the Tcl README file
    for information on this).

If you have trouble compiling Tk, I'd suggest looking at the file
"porting.notes". It contains information that people have sent me about
changes they had to make to compile Tcl in various environments. I make
no guarantees that this information is accurate, complete, or up-to-date,
but you may find it useful. If you get Tk running on a new configuration
and had to make non-trivial changes to do it, I'd be happy to receive new
information to add to "porting.notes". I'm also interested in hearing
how to change the configuration setup so that Tcl compiles on additional
platforms "out of the box".

4. Test suite

Tk now has the beginnings of a self-test suite, consisting of a set of
scripts in the subdirectory "tests". To run the test suite just type
"make test" in this directory. You should then see a printout of the
test files processed. If any errors occur, you'll see a much more
substantial printout for each error. See the README file in the
"tests" directory for more information on the test suite.

5. Getting started

Once wish is compiled you can use it to play around with the Tk
facilities. If you run wish with no arguments, it will open a small
window on the screen and read Tcl commands from standard input.
Or, you can play with some of the pre-canned scripts in the subdirectory
library/demos. See the README file in the directory for a description
of what's available. The file library/demos/widget is a script that
you can use to invoke many individual demonstrations of Tk's facilities.

If you want to start typing Tcl/Tk commands to wish, I'd suggest
starting with a widget-creation command like "button", and also learn
about the "pack" and "place" commands for geometry management. Note:
when you create a widget, it won't appear on the screen until you tell
a geometry manager about it. The only geometry managers at present
are the packer and the placer. If you don't already know Tcl, read the
Tcl book excerpt that can be FTP'ed separately from the distribution
directory.

Andrew Payne has written a very nice demo script called "The Widget Tour"
that introduces you to writing Tk scripts. This script is available
from the Tcl contributed archive described below. If you're just
getting started with Tk I strongly recommend trying out the widget tour.

6. Summary of changes in recent releases

Tk 3.6 is a minor new release that is identical to 3.4 except that it fixes a portability bug that prevents tkMain.c from compiling on some machines (R_OK isn't properly defined). Tk 3.6 should be completely compatible with both 3.4 and 3.3.

Tk 3.5 was mistake, and was withdrawn shortly after it was released.

Tk 3.4 is a minor release consisting almost entirely of bug fixes. There are no significant feature changes and Tk 3.4 should be completely compatible with Tk 3.3.

Tk 3.3 consists mostly of bug fixes plus upgrades to make it compatible with Tcl 7.0. It should not introduce any compatibility problems itself, but it requires Tcl 7.0, which introduces several incompatibilities (see the Tcl README file for details). The file "changes" contains a complete list of all changes to Tk, including both bug fixes and new features. Here is a short list of a few of the most significant new features:

1. Tk is now consistent with the book drafts. This means that the new packer syntax has been implemented and additional bitmaps and reliefs are available.

2. Tk now supports stacking order. Windows will stack in the order created, and "raise" and "lower" commands are available to change the stacking order.

3. There have been several improvements in configuration: GNU autoconf is now used for configuration; wish now supports the Tcl_AppInit procedure; and there's a patchlevel.h file that will be used for future patches. The Tk release no longer includes a Tcl release; you'll have to retrieve Tcl separately.

4. The Tk script library contains a new procedure "tk_dialog" for creating dialog boxes, and the default "tkerror" has been improved to use tk_dialog.

5. Tk now provides its own "exit" command that cleans up properly, so it's now safe to use "exit" instead of "destroy ." to end wish applications.

6. Cascade menu entries now display proper Motif arrows.

7. The main window is now a legitimate toplevel widget.

8. Wish allows prompts to be user-settable via the "tcl_prompt1" and "tcl_prompt2" variables.

7. Tcl/Tk newsgroup

There is a network news group "comp.lang.tcl" intended for the exchange of information about Tcl, Tk, and related applications. Feel free to use this newsgroup both for general information questions and for bug reports. I read the newsgroup and will attempt to fix bugs and problems reported to it.

8. Tcl/Tk contributed archive

Many people have created exciting packages and applications based on Tcl and/or Tk and made them freely available to the Tcl community. An archive of these contributions is kept on the machine harbor.ecn.purdue.edu. You can access the archive using anonymous FTP; the Tcl contributed archive is in the directory "pub/tcl".

9. Support and bug fixes

I'm very interested in receiving bug reports and suggestions for improvements. Bugs usually get fixed quickly (particularly if they are serious), but enhancements may take a while and may not happen at all unless there is widespread support for them (I'm trying to slow the rate at which Tk turns into a kitchen sink). It's becoming increasingly difficult to make incompatible changes to Tk, but it's not totally out of the question.

The Tcl/Tk community is too large for me to provide much individual support for users. If you need help I suggest that you post questions to comp.lang.tcl. I read the newsgroup and will attempt to answer esoteric questions for which no-one else is likely to know the answer. In addition, Tcl/Tk support and training are available commercially from NeoSoft. For more information, send e-mail to "info@neosoft.com".

10. Release organization

Each Tk release is identified by two numbers separated by a dot, e.g. 3.2 or 3.3. If a new release contains changes that are likely to break existing C code or Tcl scripts then the major release number increments and the minor number resets to zero: 3.0, 4.0, etc. If a new release contains only bug fixes and compatible changes, then the minor number increments without changing the major number, e.g. 3.1, 3.2, etc. If you have C code or Tcl scripts that work with release X.Y, then they should also work with any release X.Z as long as Z > Y.

Beta releases have an additional suffix of the form bx. For example, Tk 3.3b1 is the first beta release of Tk version 3.3, Tk 3.3b2 is the second beta release, and so on. A beta release is an initial version of a new release, used to fix bugs and bad features before declaring the release stable. Each new release will be preceded by one or more beta releases. I hope that lots of people will try out the beta releases and report problems back to me. I'll make new beta releases to fix the problems, until eventually there is a beta release that appears to be stable. Once this occurs I'll remove the beta suffix so that the last beta release becomes the official release.

If a new release contains incompatibilities (e.g. 4.0) then I can't promise to maintain compatibility among its beta releases. For example, release 4.0b2 may not be backward compatible with 4.0b1. I'll try to minimize incompatibilities between beta releases, but if a major problem turns up then I'll fix it even if it introduces an incompatibility. Once the official release is made then there won't be any more incompatibilities until the next release with a new major version number.

APPENDIX C

GNUPLOT 3.5 INSTALLATION MANUAL

GNUPLOT

An Interactive Plotting Program

Thomas Williams & Colin Kelley

Version 3.5 organized by: Alex Woo
Major contributors (alphabetic order):
John Campbell
Robert Cunningham
Gershon Elber
Roger Fearick
David Kotz
Ed Kubaitis
Russell Lang
Alexander Lehmann
Carsten Steger
Tom Tkacik
Jos Van der Woude
Alex Woo

Mailing list for comments: `info-gnuplot@dartmouth.edu`
Mailing list for bug reports: `bug-gnuplot@dartmouth.edu`

This manual is for GNUPLOT version 3.5

Contents

1 Gnuplot

GNUPLOT is a command-driven interactive function plotting program.

For help on any topic, type **help** followed by the name of the topic.

The new GNUPLOT user should begin by reading the **introduction** topic (type **help introduction**) and about the **plot** command (type **help plot**). Additional help can be obtained from the USENET newsgroup comp.graphics.gnuplot.

2 Copyright

```
  AUTHORS

    Original Software:
      Thomas Williams,  Colin Kelley.

    Gnuplot 2.0 additions:
        Russell Lang, Dave Kotz, John Campbell.

    Gnuplot 3.0 additions:
        Gershon Elber and many others.

  There is a mailing list for gnuplot users. Note, however, that the
  newsgroup
        comp.graphics.gnuplot
  is identical to the mailing list (they
  both carry the same set of messages). We prefer that you read the
  messages through that newsgroup, to subscribing to the mailing list.
  (If you can read that newsgroup, and are already on the mailing list,
  please send a message info-gnuplot-request@dartmouth.edu, asking to be
  removed from the mailing list.)

  The address for mailing to list members is
        info-gnuplot@dartmouth.edu
  and for mailing administrative requests is
        info-gnuplot-request@dartmouth.edu
  The mailing list for bug reports is
        bug-gnuplot@dartmouth.edu
  The list of those interested in beta-test versions is
        info-gnuplot-beta@dartmouth.edu
```

3 Introduction

GNUPLOT is a command-driven interactive function plotting program. It is case sensitive (commands and function names written in lowercase are not the same as those written in CAPS). All command names may be abbreviated. as long as the abbreviation is not ambiguous. Any number of commands may appear on a line, separated by semicolons (;). Strings are indicated with quotes. They may be either single or double quotation marks, e.g.,

```
load "filename"
cd 'dir'
```

Any command-line arguments are assumed to be names of files containing GNUPLOT commands, with the exception of standard X11 arguments, which are processed first. Each file is loaded with the **load** command, in the order specified. GNUPLOT exits after the last file is processed. When no load files are named, gnuplot enters into an interactive mode.

Commands may extend over several input lines, by ending each line but the last with a backslash (\). The backslash must be the LAST character on each line. The effect is as if the backslash and newline were not there. That is. no white space is implied, nor is a comment terminated. Therefore, commenting out a continued line comments out the entire command (see **comment**).

In this documentation. curly braces ({}) denote optional arguments to many commands, and a vertical bar (|) separates mutually exclusive choices. GNUPLOT keywords or help topics are indicated by backquotes or **boldface** (where available). Angle brackets (<>) are used to mark replaceable tokens.

For help on any topic. type **help** followed by the name of the topic.

The new GNUPLOT user should begin by reading about the **plot** command (type **help plot**).

4 Cd

The cd command changes the working directory.

Syntax:

```
cd "<directory-name>"
```

The directory name must be enclosed in quotes.

Examples:

```
cd 'subdir'
cd ".."
```

5 Clear

The **clear** command erases the current screen or output device as specified by **set output**. This usually generates a formfeed on hardcopy devices. Use **set terminal** to set the device type.

6 Command line-editing

The Unix, Atari. VMS. MS-DOS and OS/2 versions of GNUPLOT support command line-editing. Also. a history mechanism allows previous commands to be edited. and re-executed. After the command line has been edited, a newline or carriage return will enter the entire line regardless of where the cursor is positioned.

The editing commands are as follows:

Character	Function
	Line Editing
`^B`	move back a single character.
`^F`	move forward a single character.
`^A`	move to the beginning of the line.
`^E`	move to the end of the line.
`^H, DEL`	delete the previous character.
`^D`	delete the current character.
`^K`	delete from current position to the end of line.
`^L, ^R`	redraw line in case it gets trashed.
`^U`	delete the entire line.
`^W`	delete from the current word to the end of line.
	History
`^P`	move back through history.
`^N`	move forward through history.

On the IBM PC the use of a TSR program such as DOSEDIT or CED may be desired for line editing. For such a case GNUPLOT may be compiled with no line editing capability (default makefile setup). Set READLINE in the makefile and add readline.obj to the link file if GNUPLOT line editing is to be used for the IBM PC. The following arrow keys may be used on the IBM PC and Atari versions if readline is used:

Arrow key	Function
Left	same as `^B`.
Right	same as `^F`.
Ctl Left	same as `^A`.
Ctl Right	same as `^E`.
Up	same as `^P`.
Down	same as `^N`.

The Atari version of readline defines some additional key aliases:

Arrow key	Function
Undo	same as `^L`.
Home	same as `^A`.
Ctrl Home	same as `^E`.
ESC	same as `^U`.
Help	'help' plus return.
Ctrl Help	'help '.

(The readline function in gnuplot is not the same as the readline used in GNU BASH and GNU EMACS. It is somewhat compatible however.)

7 Comment

Comments are supported as follows: a # may appear in most places in a line and GNUPLOT will ignore the rest of the line. It will not have this effect inside quotes, inside numbers (including complex numbers), inside command substitutions, etc. In short, it works anywhere it makes sense to work.

8 Environment

A number of shell environment variables are understood by GNUPLOT. None of these are required, but may be useful.

If GNUTERM is defined, it is used as the name of the terminal type to be used. This overrides any terminal type sensed by GNUPLOT on start up, but is itself overridden by the .gnuplot (or equivalent) start-up file (see **start-up**), and of course by later explicit changes.

On Unix, AmigaDOS, AtariTOS, MS-DOS and OS/2, GNUHELP may be defined to be the pathname of the HELP file (gnuplot.gih).

On VMS, the symbol GNUPLOT$HELP should be defined as the name of the help library for GNUPLOT.

On Unix, HOME is used as the name of a directory to search for a .gnuplot file if none is found in the current directory. On AmigaDOS, AtariTOS, MS-DOS and OS/2, GNUPLOT is used. On VMS, SYS$LOGIN: is used. See **help start-up**.

On Unix, PAGER is used as an output filter for help messages.

On Unix, AtariTOS and AmigaDOS, SHELL is used for the **shell** command. On MS-DOS and OS/2, COMSPEC is used for the **shell** command.

On AmigaDOS, GNUFONT is used for the screen font. For example: "setenv GNUFONT sapphire/14".

On MS-DOS, if the BGI interface is used, the variable **BGI** is used to point to the full path of the BGI drivers directory. Furthermore SVGA is used to name the Super VGA BGI driver in 800x600 res., and its mode of operation as 'Name.Mode'. E.g., if the Super VGA driver is C:\TC\BGI\SVGADRV.BGI and mode 3 is used for 800x600 res., then: 'set BGI=C:\TC\BGI' and 'set SVGA=SVGADRV.3'.

9 Exit

The commands **exit** and **quit** and the END-OF-FILE character will exit GNUPLOT. All these commands will clear the output device (as the **clear** command does) before exiting.

10 Expressions

In general, any mathematical expression accepted by C, FORTRAN, Pascal, or BASIC is valid. The precedence of these operators is determined by the specifications of the C programming language. White space (spaces and tabs) is ignored inside expressions.

Complex constants may be expressed as the {<real>,<imag>}, where <real> and <imag> must be numerical constants. For example, {3,2} represents 3 + 2i; {0,1} represents i itself. The curly braces are explicitly required here.

10.1 Functions

The functions in GNUPLOT are the same as the corresponding functions in the Unix math library, except that all functions accept integer, real, and complex arguments, unless otherwise noted. The sgn function is also supported, as in BASIC.

Function	Arguments	Returns
abs(x)	any	absolute value of x, $\|x\|$; same type
abs(x)	complex	length of x, $\sqrt{\text{real}(x)^2 + \text{imag}(x)^2}$
acos(x)	any	$\cos^{-1} x$ (inverse cosine) in radians
arg(x)	complex	the phase of x in radians
asin(x)	any	$\sin^{-1} x$ (inverse sin) in radians
atan(x)	any	$\tan^{-1} x$ (inverse tangent) in radians
besj0(x)	radians	j_0 Bessel function of x
besj1(x)	radians	j_1 Bessel function of x
besy0(x)	radians	y_0 Bessel function of x
besy1(x)	radians	y_1 Bessel function of x
ceil(x)	any	$\lceil x \rceil$, smallest integer not less than x (real part)
cos(x)	radians	$\cos x$, cosine of x
cosh(x)	radians	$\cosh x$, hyperbolic cosine of x
erf(x)	any	Erf(real(x)), error function of real(x)
erfc(x)	any	Erfc(real(x)), 1.0 - error function of real(x)
exp(x)	any	e^x, exponential function of x
floor(x)	any	$\lfloor x \rfloor$, largest integer not greater than x (real part)
gamma(x)	any	Gamma(real(x)), gamma function of real(x)
ibeta(p,q,x)	any	Ibeta(real(p, q, x)), ibeta function of real(p,q,x)
inverf(x)	any	inverse error function of real(x)
igamma(a,x)	any	Igamma(real(a, x)), igamma function of real(a,x)
imag(x)	complex	imaginary part of x as a real number
invnorm(x)	any	inverse normal distribution function of real(x)
int(x)	real	integer part of x, truncated toward zero
lgamma(x)	any	Lgamma(real(x)), lgamma function of real(x)
log(x)	any	$\log_e x$, natural logarithm (base e) of x
log10(x)	any	$\log_{10} x$, logarithm (base 10) of x
norm(x)	any	normal distribution (Gaussian) function of real(x)
rand(x)	any	Rand(real(x)), pseudo random number generator
real(x)	any	real part of x
sgn(x)	any	1 if $x > 0$, -1 if $x < 0$, 0 if $x = 0$. imag(x) ignored
sin(x)	radians	$\sin x$, sine of x
sinh(x)	radians	$\sinh x$, hyperbolic sine x
sqrt(x)	any	$\sqrt{x}$, square root of x
tan(x)	radians	$\tan x$, tangent of x
tanh(x)	radians	$\tanh x$, hyperbolic tangent of x

10.2 Operators

The operators in GNUPLOT are the same as the corresponding operators in the C programming language, except that all operators accept integer, real, and complex arguments, unless otherwise noted. The ** operator (exponentiation) is supported, as in FORTRAN.

Parentheses may be used to change order of evaluation.

10.2.1 Binary

The following is a list of all the binary operators and their usages:

Binary Operators		
Symbol	Example	Explanation
**	a**b	exponentiation
*	a*b	multiplication
/	a/b	division
%	a%b	* modulo
+	a+b	addition
-	a-b	subtraction
==	a==b	equality
!=	a!=b	inequality
&	a&b	* bitwise AND
^	a^b	* bitwise exclusive OR
|	a|b	* bitwise inclusive OR
&&	a&&b	* logical AND
||	a||b	* logical OR
?:	a?b:c	* ternary operation

(*) Starred explanations indicate that the operator requires integer arguments.

Logical AND (&&) and OR (||) short-circuit the way they do in C. That is, the second && operand is not evaluated if the first is false; the second || operand is not evaluated if the first is true.

The ternary operator evaluates its first argument (a). If it is true (non-zero) the second argument (b) is evaluated and returned. otherwise the third argument (c) is evaluated and returned.

10.2.2 Unary

The following is a list of all the unary operators and their usages:

Unary Operators		
Symbol	Example	Explanation
-	-a	unary minus
~	~a	* one's complement
!	!a	* logical negation
!	a!	* factorial

(*) Starred explanations indicate that the operator requires an integer argument.

The factorial operator returns a real number to allow a greater range.

11 Help

The **help** command displays on-line help. To specify information on a particular topic use the syntax:

```
help {<topic>}
```

If <topic> is not specified. a short message is printed about GNUPLOT. After help for the requested topic is given, help for a subtopic may be requested by typing its name, extending the help request. After that subtopic has been printed. the request may be extended again, or simply pressing return goes back one level to the previous topic. Eventually. the GNUPLOT command line will return.

12 Load

The **load** command executes each line of the specified input file as if it had been typed in interactively. Files created by the **save** command can later be **loaded**. Any text file containing valid commands can be created and then executed by the **load** command. Files being **loaded** may themselves contain **load** commands. See **comment** for information about comments in commands.

The **load** command must be the last command on the line.

Syntax:

```
load "<input-file>"
```

The name of the input file must be enclosed in quotes.

Examples:

```
load 'work.gnu'
load "func.dat"
```

The **load** command is performed implicitly on any file names given as arguments to GNUPLOT. These are loaded in the order specified, and then GNUPLOT exits.

13 Pause

The **pause** command displays any text associated with the command and then waits a specified amount of time or until the carriage return is pressed. **pause** is especially useful in conjunction with **load** files.

Syntax:

```
pause <time> {"<string>"}
```

<time> may be any integer constant or expression. Choosing -1 will wait until a carriage return is hit, zero (0) won't pause at all, and a positive integer will wait the specified number of seconds.

Note: Since **pause** is not part of the plot it may interact with different device drivers differently (depending upon how text and graphics are mixed).

Examples:

```
pause -1    # Wait until a carriage return is hit
pause 3     # Wait three seconds
pause -1  "Hit return to continue"
pause 10  "Isn't this pretty?  It's a cubic-spline."
```

14 Plot

plot and **splot** are the primary commands of the program. They plot functions and data in many, many ways. **plot** is used to plot 2-d functions and data, while **splot** plots 3-d surfaces and data.

Syntax:

```
plot {ranges} {<function> | {"<datafile>" {using ...}}}
             {title} {style} {, <function> {title} {style}...}
```

```
splot {ranges} {<function> | {"<datafile>" {index i} {using ...}}}
                {title} {style} {, <function> {title} {style}...}
```

where either a <function> or the name of a data file enclosed in quotes is supplied. A function is a mathematical expression, or a pair (**plot**) or triple (**splot**) of mathematical expressions in the case of parametric functions. User-defined functions and variables may also be defined here.

plot and **splot** commands can be as simple as

```
plot sin(x)
```

and

```
splot x * y
```

or as complex as (!)

```
plot [t=1:10] [-pi:pi*2] tan(t), "data.1" using 2:3 with lines,
      t**2 with points
```

14.1 Data-file

Discrete data contained in a file can be displayed by specifying the name of the data file (enclosed in quotes) on the **plot** or **splot** command line. Data files should contain one data point per line. Lines beginning with # (or ! on VMS) will be treated as comments and ignored. For **plots**, each data point represents an (x,y) pair. For **splots**, each point is an (x,y,z) triple. For **plots** with error bars (see **plot errorbars**), each data point is either (x,y,ydelta) or (x,y,ylow,yhigh). In all cases, the numbers on each line of a data file must be separated by blank space. This blank space divides each line into columns.

For **plots** the x value may be omitted, and for **splots** the x and y values may be omitted. In either case the omitted values are assigned the current coordinate number. Coordinate numbers start at 0 and are incremented for each data point read.

To specify other formats, see **plot datafile using**.

In the **plot** command, blank lines in the data file cause a break in the plot. There will be no line drawn between the preceding and following points if the plot style is **lines** or **linespoints** (see **plot style**) This does not change the plot style, as would plotting the data as separate curves.

This example compares the data in the file population.dat to a theoretical curve:

```
pop(x) = 103*exp((1965-x)/10)
plot [1960:1990] 'population.dat', pop(x)
```

The file population.dat might contain:

```
# Gnu population in Antarctica since 1965
1965   103
1970   55
1975   34
1980   24
1985   10
```

When a data file is plotted, **samples** and **isosamples** are ignored. Curves plotted using the **plot** command are automatically extended to hold the entire curve. Similarly grid data plotted using the splot command is automatically extended, using the assumption that isolines are separated by blank lines (a line with only a CR/LF in it).

Implicitly, there are two types of 3-d datafiles. If all the isolines are of the same length, the data is assumed to be a grid data, i.e., the data has a grid topology. Cross isolines in the other parametric direction (the ith cross isoline passes through the ith point of all the provided isolines) will also be drawn for grid data. (Note contouring is available for grid data only.) If all the isolines are not of the same length, no cross isolines will be drawn and contouring that data is impossible.

For splot, data files may contain more than one mesh and by default all meshes are plotted. Meshes are separated from each other, in the file, by double blank lines. To control and splot a single mesh from a multi mesh file, use the index modifier. See **splot index** for more.

For splot if 3-d datafile and using format (see **splot datafile using**) specify only z (height field), a non parametric mode must be specified. If, on the other hand, x, y, and z are all specified, a parametric mode should be selected (see **set parametric**) since data is defining a parametric surface.

A simple example of plotting a 3-d data file is

```
        set parametric
        splot 'glass.dat'
```

or

```
        set noparametric
        splot 'datafile.dat'
```

where the file datafile.dat might contain:

```
        # The valley of the Gnu.
        10
        10
        10

        10
        5
        10

        10
        1
        10

        10
        0
        10
```

Note datafile.dat defines a 4 by 3 grid (4 rows of 3 points each). Rows are separated by blank lines.

On some computer systems with a popen function (UNIX), the datafile can be piped through a shell command by starting the file name with a '<'. For example:

```
        pop(x) = 103*exp(-x/10)
        plot '< awk "{print $1-1965, $2}" population.dat', pop(x)
```

would plot the same information as the first population example but with years since 1965 as the x axis. If you want to execute this example, you have to delete all comments from the data file above or substitute the following command for the first part of the command above (the part up to the comma):

```
        plot '< awk "$0 !~ /^#/ {print $1-1965, $2}" population.dat'
```

It is also possible to apply a single function to the "y" value only, e.g.

```
        plot 'population.dat' thru p(x)
```

For more information about 3-d plotting, see **splot**.

14.1.1 Using

The format of data within a file can be selected with the **using** option. An explicit scanf string can be used, or simpler column choices can be made.

Syntax:

```
        plot "datafile" { using { <ycol> |
                                  <xcol>:<ycol> |
                                  <xcol>:<ycol>:<ydelta> |
                                  <xcol>:<ycol>:<ylow>:<yhigh> |
                                  <xcol>:<ycol>:<ylow>:<yhigh>:<boxwidth> }
                                {"<scanf string>"} } ...
```

and

```
        splot "datafile" { using { <xcol>:<ycol>:<zcol> | <zcol> }
                                 {"<scanf string>"} } ...
```

<xcol>, <ycol>, and <zcol> explicitly select the columns to plot from a space or tab separated multicolumn data file. If only <ycol> is selected for **plot**, <xcol> defaults to 1. If only <zcol> is selected for **splot**, then only that column is read from the file. An <xcol> of 0 forces <ycol> to be plotted versus its coordinate number. <xcol>, <ycol>, and <zcol> can be entered as constants or expressions.

If errorbars (see also **plot errorbars**) are used for **plots**, ydelta (for example, a +/- error) should be provided as the third column, or ylow and yhigh as third and fourth columns.

If boxes or boxerrorbars are used for **plots**, a fifth column to specify the width of the box may be given. This implies that columns three and four must also be provided even if they are not used. If you want to plot boxes from a data file with three columns, set ylow and yhigh to y using the following command:

```
        plot "datafile" using 1:2:2:2:3 with boxes
```

Scanf strings override any <xcol>:<ycol>(:<zcol>) choices, except for ordering of input. e.g..

```
        plot "datafile" using 2:1 "%f%*f%f"
```

causes the first column to be y and the third column to be x.

If the scanf string is omitted, the default is generated based on the <xcol>:<ycol>(:<zcol>) choices. If the **using** option is omitted. "%f%f" is used for **plot** ("%f%f%f%f" for **errorbars plots**) and "%f%f%f" is used for **splot**.

Examples:

```
        plot "MyData" using "%*f%f%*20[^\n]%f" with lines
```

Data are read from the file "MyData" using the format "%*f%f%*20[^\n]%f". The meaning of this format is: "%*f" ignore the first number, "%f" then read in the second and assign to x, "%*20[^\n]" then ignore 20 non-newline characters, "%f" then read in the y value.

```
n=3;
plot "MyData", "MyData" using n
```

causes GNUPLOT to plot the second and third columns of MyData versus the first column. The command 'n=4; replot' would then plot the second and fourth columns of MyData versus the first column.

```
splot "glass.dat" using 1
```

causes GNUPLOT to plot the first coordinate of the points of glass.dat as the z coordinate while ignoring the other two coordinates.

Note: GNUPLOT first reads a line of the data file into a buffer and then does a

```
sscanf(input_buffer, scanf_string, &x, &y{, &z});
```

where 'x', 'y', and 'z' are of type 'float'. Any scanf string that specifies two (three for **splot**, three or four for **errorbars**) float numbers may be used.

14.2 Errorbars

Error bars are supported for 2-d data file plots by reading one or two additional columns specifying ydelta or ylow and yhigh respectively. No support exists for x error bars or any error bars for **splots**.

In the default situation. GNUPLOT expects to see three or four numbers on each line of the data file. either (x, y, ydelta) or (x, y, ylow, yhigh). The x coordinate must be specified. The order of the numbers must be exactly as given above. Data files in this format can easily be plotted with error bars:

```
plot "data.dat" with errorbars
```

The error bar is a vertical line plotted from (x, ylow) to (x, yhigh). If ydelta is specified instead of ylow and yhigh, ylow=y-ydelta and yhigh=y+ydelta are derived. If there are only two numbers on the line, yhigh and ylow are both set to y. To get lines plotted between the data points, **plot** the data file twice, once with errorbars and once with lines.

If y autoscaling is on. the y range will be adjusted to fit the error bars.

The **using** option may be used to specify how columns of the data file are to be assigned to x, y, ydelta, ylow, and yhigh. The x column must be provided and both the x and y columns must appear before the errorbar columns. If three column numbers are given, they are x, y, and ydelta. If four columns are given, they are x. y, ylow, and yhigh.

Examples:

```
plot "data.dat" using 1:2:3:4 with errorbars
plot "data.dat" using 3:2:6 with errorbars
plot "data.dat" using 3:4:8:7 with errorbars
```

The first example reads. x. y. ylow, and yhigh, from columns 1, 2, 3, and 4. This is equivalent to the default. The second example reads x from the third column, y from second and ydelta from the sixth column. The third example reads x from the third column, y from the fourth, ylow from the eighth. and yhigh from seventh columns.

See also **plot using** and **plot style**.

14.3 Parametric

When in parametric mode (**set parametric**) mathematical expressions must be given in pairs for **plot** and in triplets for **splot**:

```
        plot sin(t),t**2
```

or

```
        splot cos(u)*cos(v),cos(u)*sin(v),sin(u)
```

Data files are plotted as before, except any preceding parametric function must be fully specified before a data file is given as a plot. In other words, the x parametric function (sin(t) above) and the y parametric function (t**2 above) must not be interrupted with any modifiers or data functions; doing so will generate a syntax error stating that the parametric function is not fully specified.

Ranges take on a different meaning when in parametric mode. The first range on the **plot** command is the **trange**, the next is the **xrange**, and the last is the **yrange**. For **splot** the order is **urange**, **vrange**, **xrange**, **yrange**, and finally **zrange**. The following **plot** command shows setting the **trange** to [-pi:pi], the **xrange** to [-1.3:1.3] and the **yrange** to [-1:1] for the duration of the plot:

```
        plot [-pi:pi] [-1.3:1.3] [-1:1] sin(t),t**2
```

Other modifiers, such as **with** and **title**, may be specified only after the parametric function has been completed:

```
        plot sin(t),t**2 title 'Parametric example' with linespoints
```

14.4 Ranges

The optional range specifies the region of the plot that will be displayed.

Ranges may be provided on the **plot** and **splot** command line and affect only that plot, or in the **set xrange**, **set yrange**, etc., commands, to change the default ranges for future plots.

Syntax:

```
        [{<dummy-var> =} {<xmin> : <xmax>}] { [{<ymin> : <ymax>}] }
```

where <dummy-var> is the independent variable (the defaults are x and y, but this may be changed with **set dummy**) and the min and max terms can be constant expressions.

Both the min and max terms are optional. The ':' is also optional if neither a min nor a max term is specified. This allows '[ ]' to be used as a null range specification.

Specifying a range in the **plot** command line turns autoscaling for that axis off for that plot. Using one of the **set range** commands turns autoscaling off for that axis for future plots, unless changed later. (See **set autoscale**).

Examples:

This uses the current ranges:

```
        plot cos(x)
```

This sets the x range only:

```
        plot [-10:30] sin(pi*x)/(pi*x)
```

This is the same, but uses t as the dummy-variable:

```
        plot [t = -10 :30]  sin(pi*t)/(pi*t)
```

This sets both the x and y ranges:

```
        plot [-pi:pi] [-3:3]  tan(x), 1/x
```

This sets only the y range, and turns off autoscaling on both axes:

```
        plot [ ] [-2:sin(5)*-8] sin(x)**besj0(x)
```

This sets xmax and ymin only:

```
        plot [:200] [-pi:]  exp(sin(x))
```

This sets the x, y, and z ranges:

```
        splot [0:3] [1:4] [-1:1] x*y
```

14.5 Index

Splotting of multi mesh data files can be controlled via the index modifier. A data file can contain more than one mesh. and in that case all meshes in the file will be splotted by default. Meshes are separated from each other, in the data file, by double blank lines. To splot a single mesh in a multi mesh file use the index modifier which specify which mesh to splot. First mesh is mesh 0.

Example:

splot “data1” index 2 with points

will splot the third mesh in file data1 with points.

14.6 Style

Plots may be displayed in one of eight styles: **lines**, **points**, **linespoints**, **impulses**, **dots**, **errorbars**, **steps**, **boxes**, or **boxerrorbars**. The **lines** style connects adjacent points with lines. The **points** style displays a small symbol at each point. The **linespoints** style does both **lines** and **points**. The **impulses** style displays a vertical line from the x axis (or from the grid base for **splot**) to each point. The **dots** style plots a tiny dot at each point; this is useful for scatter plots with many points.

The **errorbars** style is only relevant to 2-d data file plotting. It is treated like **points** for **splots** and function **plots**. For data **plots**, **errorbars** is like **points**, except that a vertical error bar is also drawn: for each point (x,y), a line is drawn from (x,ylow) to (x,yhigh). A tic mark is placed at the ends of the error bar. The ylow and yhigh values are read from the data file's columns, as specified with the **using** option to plot. See **plot errorbars** for more information.

The **boxes** style is only relevant to 2-d plotting. Another style called **boxerrorbars** is also available and is only relevant to 2-d data file plotting. This style is a combination of the **boxes** and **errorbars** styles. The **boxes** style draws a box centred about the given x coordinate from the yaxis to the given y coordinate. The width of the box is obtained in one of three ways. First, if a data file has a fifth column, this will be used to set the width of the box. Columns 3 and 4 (for **boxerrorbars**) are necessary but ignored in this instance. Secondly, if a width has been set using the **set boxwidth** command, this will be used. Otherwise the width of each box will be calculated automatically so that it touches the adjacent boxes.

The **steps** style is only relevant to 2-d plotting. This style connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

Default styles are chosen with the **set function style** and **set data style** commands.

By default, each function and data file will use a different line type and point type, up to the maximum number of available types. All terminal drivers support at least six different point types, and re-use them, in order. if more than six are required. The LaTeX driver supplies an additional six point types (all variants of a circle). and thus will only repeat after twelve curves are plotted with points.

If desired, the style and (optionally) the line type and point type used for a curve can be specified.

Syntax:

```
        with <style> {<linetype> {<pointtype>}}
```

where <style> is either **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, or **errorbars**. The <linetype> and <pointtype> are positive integer constants or expressions and specify the line type and point type to be used for the plot. Line type 1 is the first line type used by default, line type 2 is the second line type used by default, etc.

Examples:

This plots sin(x) with impulses:

```
        plot sin(x) with impulses
```

This plots x*y with points, x**2 + y**2 default:

```
        splot x*y w points, x**2 + y**2
```

This plots tan(x) with the default function style, "data.1" with lines:

```
        plot [ ] [-2:5] tan(x), "data.1" with l
```

This plots "leastsq.dat" with impulses:

```
        plot 'leastsq.dat' w i
```

This plots the data file 'population' with boxes:

```
        plot "population" with boxes
```

This plots "exper.dat" with errorbars and lines connecting the points:

```
        plot 'exper.dat' w lines, 'exper.dat' w errorbars
```

Here 'exper.dat' should have three or four data columns.

This plots x**2 + y**2 and x**2 - y**2 with the same line type:

```
        splot x**2 + y**2 with line 1, x**2 - y**2 with line 1
```

This plots sin(x) and cos(x) with linespoints, using the same line type but different point types:

```
        plot sin(x) with linesp 1 3, cos(x) with linesp 1 4
```

This plots file "data" with points style 3:

```
        plot "data" with points 1 3
```

Note that the line style must be specified when specifying the point style, even when it is irrelevant. Here the line style is 1 and the point style is 3, and the line style is irrelevant.

See **set style** to change the default styles.

14.7 Title

A title of each plot appears in the key. By default the title is the function or file name as it appears on the plot command line. The title can be changed by using the **title** option. This option should precede any **with** option.

Syntax:

```
title "<title>"
```

where <title> is the new title of the plot and must be enclosed in quotes. The quotes will not be shown in the key.

Examples:

This plots y=x with the title 'x':

```
plot x
```

This plots the "glass.dat" file with the title 'surface of revolution':

```
splot "glass.dat" title 'surface of revolution'
```

This plots x squared with title "x^2" and "data.1" with title 'measured data':

```
plot x**2 title "x^2", "data.1" t 'measured data'
```

The title can be omitted from the key with the "notitle" option for plot and splot. This can be useful when some curves are plotted solely for decoration; for example, if one wanted a circular border for a polar plot, he could say:

Example:

```
set polar
plot my_function(x), 1 notitle
```

This would generate a key entry for "my_function" but not for "1". See the poldat.dem example.

15 Print

The **print** command prints the value of <expression> to the screen.

Syntax:

```
print <expression>
```

See **expressions**.

16 Pwd

The **pwd** command prints the name of the working directory to the screen.

Syntax:

```
pwd
```

17 Quit

The **exit** and **quit** commands and END-OF-FILE character will exit GNUPLOT. All these commands will clear the output device (as the **clear** command does) before exiting.

18 Replot

The **replot** command without arguments repeats the last **plot** or **splot** command. This can be useful for viewing a plot with different **set** options, or when generating the same plot for several devices.

Arguments specified after a **replot** command will be added onto the last **plot** (**splot**) command (with an implied ',' separator) before it is repeated. **replot** accepts the same arguments as the **plot** (**splot**) commands except that ranges cannot be specified. See **command line-editing** for ways to edit the last **plot** (**splot**) command.

19 Reread

The **reread** command causes the current gnuplot command file, as specified by a **load** command or on the command line. to be reset to its starting point before further commands are read from it. This essentially implements an endless loop of the commands from the beginning of the command file to the **reread** command. The **reread** command has no effect if input from standard input.

20 Save

The **save** command saves user-defined functions, variables, set options or all three plus the last **plot** (**splot**) command to the specified file.

Syntax:

```
        save  {<option>} "<filename>"
```

where <option> is **functions**, **variables** or **set**. If no option is used, GNUPLOT saves functions, variables, set options and the last **plot** (**splot**) command.

saved files are written in text format and may be read by the **load** command.

The filename must be enclosed in quotes.

Examples:

```
        save "work.gnu"
        save functions 'func.dat'
        save var 'var.dat'
        save set "options.dat"
```

21 Set-show

The **set** command sets LOTS of options.

The **show** command shows their settings. **show all** shows all the settings.

21.1 Angles

By default, GNUPLOT assumes the independent variable in polar plots is in units of radians. If **set angles degrees** is specified before **set polar** then the default range is [0:360] and the independent variable has units of degrees. This is particularly useful for plots of data files. The angle setting also hold for the 3-d mapping as set via the **set mapping** command.

Syntax:

```
set angles { degrees | radians }
show angles
```

21.2 Arrow

Arbitrary arrows can be placed on a plot using the **set arrow** command.

Syntax:

```
set arrow {<tag>} {from <sx>,<sy>{,<sz>}}
                  {to <ex>,<ey>{,<ez>}} {{no}head}
set noarrow {<tag>}
show arrow
```

Unspecified coordinates default to 0. The x, y, and z values are in the graph's coordinate system. The z coordinate is only used in **splot** commands. <tag> is an integer that identifies the arrow. If no tag is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or change a specific arrow. To change any attribute of an existing arrow, use the **set arrow** command with the appropriate tag, and specify the parts of the arrow to be changed. Specifying nohead requests the arrow be drawn without a head (yielding a line segment). By default, arrows have heads.

Arrows outside the plotted boundaries are permitted but may cause device errors.

Examples:

To set an arrow pointing from the origin to (1,2), use:

```
set arrow to 1,2
```

To set an arrow from (-10,4,2) to (-5,5,3), and tag the arrow number 3, use:

```
set arrow 3 from -10,4,2 to -5,5,3
```

To change the preceding arrow begin at 1,1,1, without an arrow head, use:

```
set arrow 3 from 1,1,1 nohead
```

To delete arrow number 2 use:

```
set noarrow 2
```

To delete all arrows use:

```
set noarrow
```

To show all arrows (in tag order) use:

```
show arrow
```

21.3 Autoscale

Auto scaling may be set individually on the x, y or z axis or globally on all axes. The default is to autoscale all axes.

When autoscaling, the plot range is automatically computed and the dependent axis (y for a **plot** and z for **splot**) is scaled to include the range of the function or data being plotted.

If autoscaling of the dependent axis (y or z) is not set, the current y or z range is used.

See **set yrange** or **set zrange**.

Autoscaling the independent variables (x for **plot** and x,y for **splot**) is a request to set the domain to match any data file being plotted. If there are no data files then autoscaling an independent variable has no effect. In other words, in the absence of a data file, functions alone do not affect the x range (or the y range if plotting z = f(x,y)).

See **set xrange**, or **set yrange**.

The behavior of autoscaling remains consistent in parametric mode, however, there are more dependent variables and hence more control over x, y, and z plot scales. In parametric mode, the independent or dummy variable is t for **plots** and u,v for **splots**. Autoscale in parametric mode, then, controls all ranges (t, u, v, x, y, and z) and allows x, y, and z to be fully autoscaled.

See **set parametric**.

Syntax:

```
        set autoscale <axes>
        set noautoscale <axes>
        show autoscale
```

where <axes> is either **x**, **y**, **z** or **xy**. If <axes> is not given then all axes are assumed.

Examples:

This sets autoscaling of the y axis. x axis autoscaling is not affected.

```
        set autoscale y
```

This sets autoscaling of the x and y axes.

```
        set autoscale xy
```

This sets autoscaling of the x, y and z axes.

```
        set autoscale
```

This disables autoscaling of the x, y and z axes.

```
        set noautoscale
```

This disables autoscaling of the z axis only.

```
        set noautoscale z
```

21.3.1 Parametric mode

When in parametric mode (**set parametric**) the xrange is as fully scalable as the yrange. In other words, in parametric mode the x axis can be automatically scaled to fit the range of the parametric function that is being plotted. Of course, the y axis can also be automatically scaled just as in the non-parametric case. If autoscaling on the x axis is not set, the current x range is used.

When there is a mix of data files and functions, the xrange of the functions is selected as that of the data files if autoscale is true for x. While this keeps the behavior compatible with non-parametric plotting, it may not be retained in the future. The problem is that, in parametric mode, the x and y ranges are not as distinguishable as in the non-parametric mode and this behavior may not be the most useful.

For completeness a last command **set autoscale t** is accepted. However, the effect of this "scaling" is very minor. When GNUPLOT determines that the t range would be empty it makes a small adjustment if autoscaling is true. Otherwise, GNUPLOT gives an error. Such behavior may, in fact, not be very useful and the command **set autoscale t** is certainly questionable.

splot extends the above idea similarly. If autoscaling is set then x, y, and z ranges are computed and each axis scaled to fit the resulting data.

21.4 Border

The **set border** and **set noborder** commands controls the display of the plot borders for the **plot** and **splot** commands.

Syntax:

```
        set border
        set noborder
        show border
```

21.5 Boxwidth

The set **boxwidth** command is used to set the default width of boxes in the **boxes** and **boxerrorbars** styles.

If a data file is plotted without the width being specified in the fifth column, or a function is plotted, the width of each box is set by the set **boxwidth** command. If a width is given after the set **boxwidth** command then this is used as the width. Otherwise the width of each box will be calculated automatically so that it touches the adjacent boxes.

Syntax:

```
        set boxwidth {<width>}
        show boxwidth
```

To set the box width to automatic use the command

```
        set boxwidth
```

21.6 Clabel

GNUPLOT will vary the linetype used for each contour level when clabel is set. When this option on (the default), a legend labels each linestyle with the z level it represents.

Syntax:

```
        set clabel
        set noclabel
        show clabel
```

21.7 Clip

GNUPLOT can clip data points and lines that are near the boundaries of a plot.

Syntax:

```
        set clip <clip-type>
        set noclip <clip-type>
        show clip
```

Three clip types are supported by GNUPLOT: **points**, **one**, and **two**. One, two, or all three clip types may be active for a single plot.

The **points** clip type forces GNUPLOT to clip (actually, not plot at all) data points that fall within but too close to the boundaries (this is so the large symbols used for points will not extend outside the boundary lines). Without clipping points near the boundariês may look bad; try adjusting the x and y ranges.

Setting the **one** clip type causes GNUPLOT to plot the line segments which have only one of the two endpoints within the plotting region. Only the in-range portion of the line is drawn. The alternative is to not draw any portion of the line segment.

Some lines may have both endpoints out of range, but pass through the plotting area. Setting the **two** clip-type allows the visible portion of these lines to be drawn.

In no case is a line drawn outside the plotting area.

The defaults are **noclip points**, **clip one**, and **noclip two**.

To check the state of all forms of clipping, use

```
        show clip
```

For backward compatibility with older versions, the following forms are also permitted.

```
       set clip
       set noclip
```

set clip is synonymous with set **clip points**. set **noclip** turns off all three types of clipping.

21.8 Cntrparam

Sets the different parameters for the contouring plot (see also **contour**).

Syntax:

```
 set cntrparam { { linear | cubicspline | bspline } |
    points <n> |
    order <n>  |
    levels { [ auto ] <n> |
    discrete <z1>,<z2>, ... |
    incremental {<start>, <incr>{, <end>} } }
```

Examples:

```
    set cntrparam bspline
    set cntrparam points 7
    set cntrparam order 10
    set cntrparam levels auto 5                 # 5 automatic levels
    set cntrparam levels discrete .1,1/exp(1),.9  # 3 discrete at .1,.37,.9
    set cntrparam levels incremental  0,.1,.4
    # 5 incremental levels at 0, .1, .2, .3 and .4
    set cntrparam levels 10
    # sets n = 10 retaining current setting of auto, discr. and
    # increment's start and increment value, while changing end
    set cntrparam levels incremental 100,50
    # set start = 100 and increment = 50, retaining n levels
```

This command controls the way contours are plotted. <n> should be an integral constant expression and <z1>, <z2> any constant expressions. The parameters are:

linear, cubicspline, bspline - Controls type of approximation or interpolation. If **linear**, then the contours are drawn piecewise linear, as extracted from the surface directly. If **cubicspline**, then piecewise linear contours are interpolated to form a somewhat smoother contours, but which may undulate. The third option is the uniform **bspline**, which only approximates the piecewise linear data but is guaranteed to be smoother.

points - Eventually all drawings are done with piecewise linear strokes. This number controls the number of points used to approximate a curve. Relevant for **cubicspline** and **bspline** modes only.

order - Order of the bspline approximation to be used. The bigger this order is, the smoother the resulting contour. (Of course, higher order bspline curves will move further away from the original piecewise linear data.) This option is relevant for **bspline** mode only. Allowed values are integers in the range from 2 (linear) to 10.

levels - Number of contour levels, 'n'. Selection of the levels is controlled by 'auto' (default), 'discrete', and 'incremental'. For 'auto', if the surface is bounded by zmin and zmax then contours will be generated from zmin+dz to zmax-dz in steps of size dz, where dz = (zmax - zmin) / (levels + 1). For 'discrete', contours will be generated at z = z1, z2 ... as specified. The number of discrete levels is limited to MAX_DISCRETE_LEVELS, defined in plot.h to be 30. If 'incremental', contours are generated at <n> values of z beginning at <start> and increasing by <increment>.

21.9 Contour

Enable contour drawing for surfaces. This option is available for **splot** only.

Syntax:

```
        set contour { base | surface | both }
        set nocontour
```

If no option is provided to **set contour**, the default is **base**. The three options specify where to draw the contours: **base** draws the contours on the grid base where the x/ytics are placed, **surface** draws the contours on the surfaces themselves, and **both** draws the contours on both the base and the surface.

See also **set cntrparam** for the parameters that affect the drawing of contours.

21.10 Data style

The **set data style** command changes the default plotting style for data plots.

Syntax:

```
        set data style
        show data style
        set data style <style-choice>
```

In the first case, **set data style** returns the possible style choices: **lines**, **points**, **linespoints**, **dots**, **steps**, **impulses**, **errorbars**, **boxes** or **boxerrorbars**. **show data style** shows the current default plotting style for data. **set data style dots** would actually change the default plotting style. See also **plot**.

21.11 Dgrid3d

Enables and sets the different parameters for non grid to grid data mapping.

Syntax:

```
set dgrid3d {,{<row_size>}{,{<col_size>}{,<norm>}}}
set nodgrid3d
```

Examples:

```
set dgrid3d 10,10,2
set dgrid3d ,,4
```

The first selects a grid of size 10 by 10 to be constructed and the use of L2 norm in the distance computation. The second only modifies the norm to be used to L4.

By default this option is disabled. When enabled, 3d data read from a file is always treaded as a scattered data set. A grid with dimensions derived from a bounding box of the scattered data and size as specified by the row/col_size above is created for plotting and contouring. The grid is equally spaced in x and y while the z value is computed as a weighted average of the scattered points distance to the grid points. The closer the scatter points to a grid point are the more effect they have on that grid point. The third, norm, parameter controls the "meaning" of the distance, by specifying the distance norm. This distance computation is optimized for powers of 2 norms, specifically 1, 2, 4, 8, and 16, but any nonnegative integer can be used.

This dgrid3d option is a simple low pass filter that converts scattered data to a grid data set. More sophisticated approaches to this problem exists and should be used as a preprocess to and outside gnuplot if this simple solution is found inadequate.

21.12 Dummy

By default, GNUPLOT assumes that the independent variable for the **plot** command is x, and the independent variables for the **splot** command are x and y. They are called the dummy variables because it is just a notation to indicate the independent variables. The **set dummy** command changes these default dummy variable names. For example, it may be more convenient to call the dummy variable t when plotting time functions:

```
set dummy t
plot sin(t), cos(t)
```

Syntax:

```
set dummy <dummy-var>{,<dummy-var>}
show dummy
```

Examples:

```
set dummy u,v
set dummy ,s
```

to set both dummy variables to u and v or set only the second variable to s.

The set **parametric** command also changes the dummy variables (to t for **plot** and u,v for **splots**).

21.13 Format

The format of the tic-mark labels can be set with the set **format** command. The default format for both axes is "%g". but other formats such as "%.2f" or "%3.0fm" are often desirable. Anything accepted by printf when given a double precision number, and then accepted by the terminal, will work. In particular, the formats f, e, and g will work, and the d, o, x, c, s, and u formats will not work.

Syntax:

```
        set format {<axes>} {"<format-string>"}
        show format
```

where <axes> is either **x**, **y**, **z**, **xy**, or nothing (which is the same as **xy**). The length of the string representing a ticmark (after formatting with printf) is restricted to 100 characters. If the format string is omitted, the format will be returned to the default "%g". For LaTeX users, the format "$%g$" is often desirable. If the empty string "" is used, no label will be plotted with each tic, though the tic mark will still be plotted. To eliminate all tic marks, use set **noxtics** or set **noytics**.

See also set **xtics** and set **ytics** for more control over tic labels.

21.14 Function style

The set **function style** command changes the default plotting style for functions.

Syntax:

```
        set function style
        show function style
        set function style <style-choice>
```

In the first case, set **function style** returns the possible style choices: **lines**, **points**, **linespoints**, **dots**, **steps**, **impulses**, **errorbars**, **boxes**, or **boxerrorbars**. **show function style** shows the current default plotting style for functions. set **function style linespoints** would actually change the default plotting style. See also **plot**.

21.15 Functions

The **show functions** command lists all user-defined functions and their definitions.

Syntax:

```
        show functions
```

21.16 Grid

The optional set **grid** draws a grid at the tic marks with the axis linetype.

Syntax:

```
        set grid
        set nogrid
        show grid
```

21.17 Hidden3d

The **set hidden3d** command enables hidden line removal for explicit surface plotting (see **splot**). Hidden line removal may be used for both explicit functions and for explicit data. It now works for parametric surfaces as well.

When this flag is set both the surface hidden portion and possibly its hidden contours (see **set contour**) as well as the hidden grid will be removed. Labels and arrows are always visible and are unaffected by this command.

Each surface has its hidden parts removed with respect to itself and to other surfaces, if more than one surface is plotted. This mode is meaningful when surfaces are plotted using line style drawing only.

Syntax:

```
        set hidden3d
        set nohidden3d
        show hidden3d
```

21.18 Isosamples

An isoline is a curve parametrized by one of the surface parameters while the other surface parameter is fixed. Isolines are a simple means to display a surface. By fixing the u parameter of surface s(u,v), the iso-u lines of the form c(v) = s(u0,v) are produced, and by fixing the v parameter. the iso-v lines of the form c(u) = s(u,v0) are produced.

The isoline density of surfaces may be changed by the **set isosamples** command. By default, sampling is set to 10 isolines per u or v axis. A higher sampling rate will produce more accurate plots, but will take longer. This parameter has no effect on data file plotting.

Syntax:

```
        set isosamples <iso_1> {,<iso_2>}
        show isosamples
```

Each surface plot will have <iso_1> iso-u lines and <iso_2> iso-v lines. If you only specify <iso_1>, <iso_2> will be set to the same value as <iso_1>.

When a surface plot is being done without the removal of hidden lines, **set samples** also has an effect on the number of points being evaluated. See **set samples**.

21.19 Key

The **set key** enables a key describing curves on a plot. By default the key is placed in the upper right corner of the plot.

Syntax:

```
        set key
        set key <x>,<y>{,<z>}
        set nokey
        show key
```

The coordinates <x>. <y> (and <z> for **splots**) specify the location of the key on the plot. The key is drawn as a sequence of lines. with one plot described on each line. On the right hand side of each line is a representation that attempts to mimic the way the curve is plotted. On the left side of each line is the text description. obtained from the **plot** command. See **plot title** to change this description. The lines are vertically arranged so an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of this line that are specified with the **set key** command. In a **plot**, only the x and y coordinates are used to specify the line position. For a **splot**, x, y and z are all being used as a 3-d location mapped using the same mapping as the plot itself to form the required 2-d screen position of the imaginary line.

Some or all of the key may be outside of the plot boundary, although this may interfere with other labels and may cause an error on some devices.

Examples:

This places the key at the default location:

```
        set key
```

This disables the key:

```
        set nokey
```

This places a key at coordinates 2,3.5,2

```
        set key 2,3.5,2
```

21.20 Label

Arbitrary labels can be placed on the plot using the **set label** command. If the z coordinate is given on a **plot** it is ignored; if it is missing on a **splot** it is assumed to be 0.

Syntax:

```
         set label {<tag>} {"<label_text>"} {at <x>,<y>{,<z>}}
                           {<justification>}
         set nolabel {<tag>}
         show label
```

The text defaults to "". and the position to 0,0,0. The <x>, <y>, and <z> values are in the graph's coordinate system. The tag is an integer that is used to identify the label. If no <tag> is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or change a specific label. To change any attribute of an existing label, use the **set label** command with the appropriate tag, and specify the parts of the label to be changed.

By default, the text is placed flush left against the point x,y,z. To adjust the way the label is positioned with respect to the point x,y,z, add the parameter <justification>, which may be **left**, **right** or **center**, indicating that the point is to be at the left, right or center of the text. Labels outside the plotted boundaries are permitted but may interfere with axes labels or other text.

Examples:

To set a label at (1,2) to "y=x" use:

```
         set label "y=x" at 1,2
```

To set a label "y=x^2" with the right of the text at (2,3,4), and tag the label number 3, use:

```
         set label 3 "y=x^2" at 2,3,4 right
```

To change the preceding label to center justification, use:

```
         set label 3 center
```

To delete label number 2 use:

```
set nolabel 2
```

To delete all labels use:

```
set nolabel
```

To show all labels (in tag order) use:

```
show label
```

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

21.21 Logscale

Log scaling may be set on the x, y, and z axes.

Syntax:

```
set logscale <axes> <base>
set nologscale <axes>
show logscale
```

where <axes> may be any combinations of **x**, **y**, and **z**, in any order, and where <base> is the base of the log scaling. If <base> is not given, then 10 is assumed. If <axes> is not given then all three axes are assumed. The command **set logscale** turns on log scaling on the specified axes, while **set nologscale** turns off log scaling.

Examples:

To enable log scaling in both x and z axes:

```
set logscale xz
```

To enable scaling log base 2 of the y axis:

```
set logscale y 2
```

To disable z axis log scaling:

```
set nologscale z
```

21.22 Mapping

Syntax:

```
set mapping { cartesian | spherical | cylindrical }
```

Data for **splots** are usually in regular Euclidean space and are provided in Cartesian coordinates. Such 3-d data require three coordinates (x, y and z) or one coordinate (only z) in each line in the data file. In order to be able to use spherical or cylindrical coordinate systems, use the **set mapping** command. In both cases two coordinates are expected in each line of the data. For a spherical coordinate system. these are theta and phi (in units as specified by **set angles**) and the mapping is:

```
        x = cos( theta ) * cos( phi )
        y = sin( theta ) * cos( phi )
        z = sin( phi )
```

For a cylindrical coordinate system, the mapping uses two variables, theta (in units as specified by set angles) and z:

```
        x = cos( theta )
        y = sin( theta )
        z = z
```

Again, note that mapping will affect data file splots only.

21.23 Offsets

The amount of the graph that the plot takes up may be controlled to some extent with the set **offsets** command. This command takes four offset arguments: <left>, <right>, <top> and <bottom>. By default, each offset is 0. Each offset may be a constant or an expression. Left and right offsets are given in units of the x axis, while top and bottom offsets are given in units of the y axis. The plot of sin(x), displayed with offsets of 0, 0, 2, 2 will take up 1/3 of the displayed y axis. Offsets are particularly useful with polar coordinates as a means of compensating for aspect ratio distortion. Offsets are ignored in **splots**.

Syntax:

```
        set offsets <left>, <right>, <top>, <bottom>
        show offsets
```

21.24 Output

By default, plots are displayed to the standard output. The set **output** command redirects the display to the specified file or device.

Syntax:

```
        set output {"<filename>"}
        show output
```

The filename must be enclosed in quotes. If the filename is omitted, output will be sent to the standard output.

On machines with popen functions (UNIX), output can be piped through a shell command if the first letter of the filename is '|'. For instance,

Syntax:

```
        set output "|lpr -Plaser filename"
        set output "|lp -dlaser filename"
```

(On MSDOS machines, set output "prn" will direct the output to the default printer.)

21.25 Parametric

The set **parametric** command changes the meaning of **plot** (**splot**) from normal functions to parametric functions. The command set **noparametric** changes the plotting style back to normal, single-valued expression plotting.

In 2-d plotting, a parametric function is determined by a pair of parametric functions operating on a parameter. An example of a 2-d parametric function would be plot sin(t),cos(t) (which defines a circle).

For 3-d plotting, the surface is described as x=f(u,v), y=g(u,v), z=h(u,v). Therefore a triplet of functions are required. An example of 3-d parametric function would be cos(u)*cos(v),cos(u)*sin(v),sin(u) (which defines a sphere). It takes three parametric function specifications in terms of the parametric dummy arguments to describe a single graph.

The total set of possible plots is a superset of the simple f(x) style plots, since the two (three) functions can describe the x and y (and z) values to be computed separately. In fact, plots of the type t,f(t) (u,v,f(u,v)) are equivalent to those produced with f(x) when the x values are computed using the identity function as the first function.

Note that the order the parametric functions are specified is xfunction, yfunction (and zfunction) and that each operates over the common parametric domain.

Also, the set **parametric** function implies a new range of values. Whereas the normal f(x) and f(x,y) style plotting assume an xrange and yrange (and zrange), the parametric mode additionally specifies a trange, urange, and vrange. These ranges may be set directly with **set trange**, **set urange** and **set vrange**, or by specifying the range on the **plot** or **splot** commands. Currently the default range for these parametric variables is [-5:5]. Setting the ranges to something more meaningful is expected.

21.26 Polar

The set **polar** command changes the meaning of the plot from rectangular coordinates to polar coordinates. In polar coordinates, the dummy variable (x) is an angle. The range of this angle is changed from whatever it was to [0:2*pi], or, if degree unit has been selected, to [0:360] (see **set angles**).

The command **set nopolar** changes the meaning of the plot back to the default rectangular coordinate system. The range of x is changed from whatever it was to [-10:10].

The set **polar** command is not supported for **splots**. See the set **mapping** command for similar functionality for **splots**.

While in polar coordinates the meaning of an expression in x is really r = f(x), where x is an angle of rotation. The xrange controls the domain (the angle) of the function, and the yrange controls the range (the radius). The plot is plotted in a rectangular box, and the x and y axes are both in units of the radius. Thus, the yrange controls both dimensions of the plot output. The tics and units are written along the axes rather than at the left and bottom. These unit are offset by <rmin> specified by the **rrange** (See **set rrange**). It is not possible to specify different output dimensions in the x or y directions. The yrange can be used to shift the plot diagonally to display only the first or third quadrants.

Syntax:

```
        set polar
        set nopolar
        show polar
```

Example:

```
        set polar
        plot x*sin(x)
        plot [-2*pi:2*pi] [-3:3] x*sin(x)
```

The first plot uses the default polar angular domain of 0 to 2*pi. The radius (and the size of the plot) is scaled automatically. The second plot expands the domain, and restricts the range of the radius (and the size of the plot) to [-3:3].

21.27 Rrange

The **set rrange** command sets the radial range used to compute x and y values when in polar mode. If not in polar mode (see **set polar**) then this range is not used. Use of this command offsets the polar singularity to the <rmin> value and shifts the units on the axes tic marks. For instance, **set rrange [-40:40]** would set the origin to -40 and would plot values of radial values between -40 to 40. Thus, if 360 degrees of data were plotted, then the plot would extend 80 units in radially from the origin. To view the entire plot, a **set yrange [-80:80]** command would create a square viewport with a circular plot tangent at the axes. Because **xrange** is used specify the angular extent, only a square viewport can be specified by **yrange**. For instance, **set yrange [0:80]** would display the first quadrant and **set yrange [-80:0]** would display the third quadrant. Any square viewport of any size can be specified but it is constrained to be centered on a 45 degree line.

This range may also be specified on the **plot** command line when in polar mode.

Syntax:

```
        set rrange [{<rmin> : <rmax>}]
```

where <rmin> and <rmax> terms are constants or expressions.

Both the <rmin> and <rmax> terms are optional. Anything omitted will not be changed, so

```
        set rrange [:10]
```

changes rmax to 10 without affecting rmin.

21.28 Samples

The sampling rate of functions may be changed by the **set samples** command. By default, sampling is set to 100 points. A higher sampling rate will produce more accurate plots, but will take longer. This parameter no longer has any effect on data-file plotting.

Syntax:

```
        set samples <samples_1> {,<samples_2>}
        show samples
```

When a 2-d plot is being done, only the value of <samples_1> is relevant.

When a surface plot is being done without the removal of hidden lines, the value of samples specifies the number of samples that are evaluated per iso line. Each iso-v line will have <sample_1> samples and each iso-u line will have <sample_2> samples. If you only specify <samples_1>, <samples_2> will be set to the same value as <samples_1>. See also **set isosamples**.

21.29 Size

The **set size** command scales the displayed size of the plot. On some terminals, changing the size of the plot will result in text being misplaced. Increasing the size of the plot may produce strange results. Decreasing is safer.

Syntax:

```
        set size {<xscale>,<yscale>}
        show size
```

The <xscale> and <yscale> values are the scaling factors for the size. The defaults (1,1) are selected if the scaling factors are omitted.

Examples:

To set the size to normal size use:

```
        set size
```

To make the plot half size use:

```
        set size 0.5,0.5
```

To make a landscape plot have a 1:1 aspect ratio in polar mode use:

```
        set size 0.721,1.0
```

To show the size use:

```
        show size
```

For the LaTeX and Fig terminals the default size (scale factor 1,1) is 5 inches wide by 3 inches high. The big Fig terminal (**bfig**) is 7 inches wide by 5 inches high. The postscript default is landscape mode 10 inches wide and 7 inches high. Note that the size of the plot includes the space used by the labels; the plotting area itself is smaller.

21.30 Style

Plots may be displayed in one of eight styles: **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, **errorbars**, **boxes**. or **boxerrorbars**. The **lines** style connects adjacent points with lines. The **points** style displays a small symbol at each point. The **linespoints** style does both **lines** and **points**. The **impulses** style displays a vertical line from the x axis (or from the grid base for **splot**) to each point. The **dots** style plots a tiny dot at each point; this is useful for scatter plots with many points.

The errorbars style is only relevant to 2-d data file plotting. It is treated like **points** for **splots** and function plots. For data plots, errorbars is like **points**, except that a vertical error bar is also drawn: for each point (x,y), a line is drawn from (x,ylow) to (x,yhigh). A tic mark is placed at the ends of the error bar. The ylow and yhigh values are read from the data file's columns, as specified with the **using** option to plot. See **plot errorbars** for more information.

The boxes style is only relevant to 2-d plotting. It draws a box centred about the given x coordinate from the yaxis to the given y coordinate. The width of the box is obtained in one of three ways. If a data file has a fifth column, this will be used to set the width of the box. Otherwise, if a width has [illegible] this will be used. Otherwise the width of each box will [illegible] adjacent boxes. [illegible] also available and is only relevant to 2-d data file plotting. This style is a combination of the boxes and **errorbars** styles.

The **steps** style is only relevant to 2-d plotting. This style connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

Default styles are chosen with the **set function style** and **set data style** commands. See **plot style** for information about how to override the default plotting style for individual functions.

Syntax.

```
        set function style <style>
         set data style <style>
          [illegible] function style
            [illegible] style
```

where <style> is **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, **errorbars**, **boxes**, or **boxerrorbars**.

21.31 Surface

set surface controls the display of surfaces. It is useful if contours are to be displayed by themselves. Whenever **set nosurface** is issued, no surface isolines/mesh will be drawn. See also **set contour**.

Syntax:

```
        set surface
        set nosurface
        show surface
```

21.32 Terminal

GNUPLOT supports many different graphics devices. Use the **set terminal** command to select the type of device for which GNUPLOT will produce output.

Syntax:

```
        set terminal {<terminal-type>}
        show terminal
```

If <terminal-type> is omitted, GNUPLOT will list the available terminal types. <terminal-type> may be abbreviated.

Use **set output** to redirect this output to a file or device.

Several terminals have additional options. For example, see **dumb**, **iris4d**, **hpljii** or **postscript**.

21.32.1 Aifm

Several options may be set in the Adobe Illustrator 3.0 driver.

Syntax:

```
        set terminal aifm {<color>}
                               {"<fontname>"} {<fontsize>}
```

Selecting default sets all options to their default values. <color> is either **color** or **monochrome**. "<fontname>" is the name of a valid PostScript font. <fontsize> is the size of the font in PostScript points, before scaling by the **set size** command. Defaults are **monochrome**, "Helvetica", and 14pt.

Also, since AI does not really support multiple pages, multiple graphs will be output directly on one another. However, each graph will be grouped individually, making it easy to separate them inside AI (just pick them up and move them).

Examples:

```
        set term aifm
        set term aifm 22
        set size 0.7,1.4
        set term aifm color "Times-Roman" 14
```

21.32.2 Atari ST

The **atari** terminal has an option to set the character size and the screen colors. The driver expects a space separated list the char size and maximal 16 3 digit hex numbers where each digit represents RED, GREEN and BLUE (in that order). The range of 0-15 is scaled to whatever color range the screen actually has. On a normal ST screen, odd and even intensities are the same.

Examples:

```
set terminal atari 4 # (use small (6x6) font)
set terminal atari 6 0 # (set monochrome screen to white on black)
set terminal atari 13 0 fff f00 f0 f ff f0f ff0
# (set first eight colors to black, white, green, blue, cyan, \
  purple, and yellow and use large font (8x16).)
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence.

21.32.3 Dumb

The dumb terminal driver has an optional size specification.

Syntax:

```
set terminal dumb {<xsize> <ysize>}
```

where <xsize> and <ysize> set the size of the dumb terminals. Default is 79 by 24.

Examples:

```
set term dumb
set term dumb 79 49 # VGA screen--why would anyone want to do that?
```

21.32.4 Epson

This set of drivers support Epson printers and derivatives. See also the NEC driver. **epson** is a generic 9 wire printer with a resolution of 512x384. **starc** is a Star Color printer with the same resolution. **epson180** and **epson60** are 180 dpi and 60 dpi drivers for newer 24 wire printers. This also includes bubble jet printers. Their resolutions are 1260x1080 and 480x360, respectively. The **tandy60** is identical to the **epson60** driver with one additional escape sequence to start IBM mode. With all of these drivers, a binary copy is required on a PC to print. Do not use **print**.

```
copy file /b lpt1:
```

21.32.5 Gpic

This driver is only known to work the Free Software Foundation gpic/groff package. Modification for the Document Workbench pic/troff package would be appreciated. FSF gpic can also produce TeX output.

A simple graph can be formatted using

```
groff -p -mpic -Tps file.pic > file.ps.
```

The output from pic can be pipe-lined into eqn, so it is possible to put complex functions in a graph with the set label and set {x/y}label commands. For instance,

```
set ylab '@space 0 int from 0 to x alpha ( t ) roman d t@'
```

Will label the y-axis with a nice integral if formatted with the command:

```
gpic filename.pic | geqn -d@@ -Tps | groff -m[macro-package] -Tps
     > filename.ps
```

Figures made this way can be scaled to fit into a document. The pic language is easy to understand, so the graphs can be edited by hand if need be. All coordinates in the pic-file produced by gnuplot are given as x+gnuplotx and y+gnuploty. By default x and y are given the value 0 If this line is removed with an editor in a number of files one can put several graphs i one figure like this (default size is 5.0x3.0 inches)

```
.PS 8.0
x=0;y=3
copy "figa.pic"
x=5;y=3
copy "figb.pic"
x=0;y=0
copy "figc.pic"
x=5;y=0
copy "figd.pic"
.PE
```

This will produce an 8 inches wide figure with four graphs in two rows on top of each other

One can also achieve the same thing by the command

```
set term pic x y
```

For example, using

```
.PS 6.0
copy "trig.pic"
.PE
```

21.32.6 Hpljii

The HP LaserJet II and HP DeskJet drivers have a single option.

Syntax:

```
      set terminal hpljii {<resolution>}
      set terminal hpdj   {<resolution>}
```

where <resolution> is the resolution of the output in dots per inch. It must be 75, 100, 150 or 300. Note: there must be enough memory available to rasterize at the higher resolutions.

Example:

```
      set terminal hpljii 150
```

21.32.7 Latex

The LaTeX and EMTeX driver allows one to specify a font type and a font size for the labels around a gnuplot graph.

Options are: Fonts:

```
      default         (Roman 10 point)
      courier
      roman
```

at any size you specify. (BEWARE METAFONT will not like odd sizes.) eg.

```
      gnuplot> set term latex courier 5
```

Unless your driver is capable of building fonts at any size (e.g. dvips), stick to the standard 10, 11 and 12 point size.

21.32.8 Iris4d

The iris4d driver can operate in two modes.

Syntax:

```
        set terminal iris4d {24}
```

If the hardware supports only 8 bits, use the default **set terminal iris4d**. If, however, the hardware supports 24 bits (8 per red/green/blue), use **set terminal iris4d 24**.

When using 24-bit mode, the colors can be directly specified via the file .gnuplot_iris4d that is searched in the current directory and then in the home directory specified by the HOME environment variable. This file holds RGB values for the background, border, labels and nine plotting colors, in that order. For example, here is a file containing the default colors:

```
        85   85   85     /* Back Ground */
        0    0    0      /* Boundary */
        170  0    170    /* Labeling */
        85   255  255    /* Plot Color 1 */
        170  0    0      /* Plot Color 2 */
        0    170  0      /* Plot Color 3 */
        255  85   255    /* Plot Color 4 */
        255  255  85     /* Plot Color 5 */
        255  85   85     /* Plot Color 6 */
        85   255  85     /* Plot Color 7 */
        0    170  170    /* Plot Color 8 */
        170  170  0      /* Plot Color 9 */
```

This file has exactly 12 lines of RGB triples. No empty lines are allowed and anything after the third number in line is ignored.

21.32.9 Mf

The mf terminal driver creates a input file to the MetaFont program. Thus a figure may be used in the TeX document in the same way as a character is.

To use the plot in a document the MetaFont program must be run with the output file from GnuPlot as input. Thus, the user needs a basic knowledge of the font creating process and inclusion of a new font in a document. However. if the Metafont program is set up properly at the local site an unexperienced user could perform the operation without much trouble.

The text support is based on a MetaFont character set. Currently the Computer Modern Roman font set is input but the user are in principal free to chose whatever fonts he/she needs. The MetaFont source files for the chosen font must be available. Each character is stored in a separate picture variable in MetaFont. These variables may be manipulated (rotated, scaled etc.) when characters are needed. The drawback is the interpretation time in the MetaFont program. On some machines (i.e. PC) the limited amount of memory available may also cause problem if too many pictures are stored.

Metafont Instructions - Set your terminal to metafont:

```
  set terminal mf
```

- Select an output-file, e.g.:

```
  set output "myfigures.mf"
```

- Do your plots. Each plot will generate a separate character. Its default size will be 5*3 inches. You can change the size by saying set size 0.5,0.5 or whatever fraction of the default size you want to have.

- Quit gnuplot.

- Generate a tfm- and gf-file by running metafont on the output of gnuplot. Since the plot is quite large (5*3 in), you will have to use a version of metafont that has a value of at least 150000 for memmax. On Unix-systems these are conventionally installed under the name bigmf. For the following assume that the command virmf stands for a big version of metafont. For example:

- Invoke metafont:

```
    virmf '&plain'
```

- Select the output device: At the metafont prompt ('*') type:

```
    \mode:=CanonCX;      % or whatever printer you use
```

- Optionally select a magnification:

```
    mag:=1;              % or whatever you wish
```

- Input the gnuplot-file:

```
    input myfigures.mf
```

On a typical Unix machine there will usually be a script called mf that executes virmf '&plain', so you probably can substitute mf for virmf &plain. This will generate two files: mfput.tfm and mfput.$$$gf (where $$$ indicates the resolution of your device). The above can be conveniently achieved by typing everything on the command line. e.g.: virmf '&plain' '\mode:=CanonCX; mag:=1; input myfigures.mf' In this case the output files will be named myfigures.tfm and myfigures.300gf.

- Generate a pk-file from the gf-file using gftopk:

```
  gftopk myfigures.300gf myfigures.300pk
```

The name of the output-file for gftopk depends on the dvi-driver you use. Ask your local TeX-administrator about the naming conventions. Next, either install the tfm- and pk-files in the appropriate directories. or set your environment-variables properly. Usually this involves setting TEXFONTS to include the current directory and do the same thing for the environment-variable that your dvi-driver uses (no standard name here...). This step is necessary so that TeX will find the font-metric file and your dvi-driver will find the pk-file.

- To include your plots in your document you have to tell TeX the font:

```
\font\gnufigs=myfigures
```

Each plot you made is stored in a single character. The first plot is character 0, the second is character 1, and so on... After doing the above step you can use the plots just like any other characters. Therefore, to place plots 1 and 2 centered in your document, all you have to do is:

```
\centerline{\gnufigs\char0}
\centerline{\gnufigs\char1}
```

in plain TeX. For LaTeX you can, of course, use the picture environment and place the plot according to your wishes using the \makebox and \put macros.

It saves you a lot of time, once you have generated the font, since TeX handles the plots as characters and uses minimal time to place them. Also the documents you make change more often, than the plots do. Also it saves a lot of TeX-memory. One last advantage of using the metafont-driver is that the dvi-file really remains device independent, because no \special-commands are used as in the eepic- and tpic-drivers.

21.32.10 Mif

Several options may be set in the MIF 3.00 driver.

Syntax:

```
     set terminal mif {<pentype>} {<curvetype>} {<help>}
```

<pentype> selects "colour" of the graphics.

```
     'colour'     plot lines with line types >= 0 in colour (MIF sep. 2-7).
     'monochrome' plot all line types in black (MIF sep. 0).
```

<curvetype> selects how "curves" are plotted.

```
     'polyline'   plot curves as continuous curves.
     'vectors'    plot curves as collections of vectors
```

<help> print online help on standard error output.

```
     'help'       print a short description of the usage, and the options
     '?'          print a short description of the usage
```

This terminal driver produces Frame Maker MIF format version 3.00. It plots in MIF Frames with the size 15*10 [cm]. and plot primitives with the same pen will be grouped in the same MIF group. Plot primitives in a gnuplot plot will be plotted in a MIF Frame, and several MIF Frames are collected in one large MIF Frame. Plot primitives with line types >= 0 will as default be drawn in colour. As default curves are plotted as continuous lines. The MIF font used for text is "Times".

Examples:

```
set term mif
set term mif vectors
set term mif help
```

21.32.11 Nec-cp6

One option may be set in the nec-cp6 driver. The resolution of this driver is 400x320.

Syntax:

```
    set terminal nec-cp6 monochrome
    set terminal nec-cp6 color
    set terminal nec-cp6 draft
```

21.32.12 Pbm

Several options may be set in the PBMplus driver.

Syntax:

```
    set terminal pbm {<fontsize>} {<colormode>}
```

where <fontsize> is **small**, **medium**, or **large** and <colormode> is **monochrome**, **gray** or **color**. Default size is 640 pixels wide and 480 pixels high. The output for **monochrome** is a portable bitmap (one bit per pixel). The output for **gray** is a portable graymap (three bits per pixel). The output for **color** is a portable pixmap (color, four bits per pixel). The output of these drivers can be used with Jef Poskanzer's excellent PBMPLUS package which provides programs to convert the above PBMPLUS formats to GIF, TIFF, MacPaint, Macintosh PICT, PCX, X11 bitmap and many others.

Examples:

```
    set term pbm small
    set size 2,2
    set term pbm color medium
```

21.32.13 Pcl5

Three options may be set in the pcl5 driver. The driver actually uses HPGL-2 but there is a name conflict among the terminal devices.

Syntax:

```
    set terminal pcl5 {<mode>} {<font>} {<fontsize>}
```

where <mode> is **landscape.** or **portrait**, <font> is **stick**, **univers**, or **cg_times**, and fontsize is the size in points.

```
    set terminal pcl5 landscape
```

21.32.14 Postscript

Several options may be set in the PostScript driver.

Syntax:

```
    set terminal postscript {<mode>} {<color>} {<dashed>}
                            {"<fontname>"} {<fontsize>}
```

where <mode> is **landscape**, **portrait**, **eps** or **default**. Selecting default sets all options to their defaults. <color> is either **color** or **monochrome**. <dashed> is either **solid** or **dashed**. "<fontname>" is the name of a valid PostScript font. <fontsize> is the size of the font in PostScript points. Defaults are **landscape**, **monochrome**, **dashed**, "Helvetica", and 14pt. Default size of PostScript plot is landscape mode 10 inches wide and 7 inches high.

To get EPS output, use the **eps** mode and make only one plot per file. In **eps** mode the whole plot is halved in size; the fonts are half the given size, and the plot is 5 inches wide and 3.5 inches high.

Examples:

```
set term postscript default       # old postscript
set term postscript landscape 22  # old psbig
set term postscript eps 14   # old epsf1
set term postscript eps 22   # old epsf2
set size 0.7,1.4
set term post portrait color "Times-Roman" 14
```

21.32.15 Regis

The **regis** terminal device has the option of using 4 or 16 colors. The default is 4. For example:

```
set term regis 16
```

21.32.16 Table

Instead of producing a picture, term type **table** prints out the evaluation results in a multicolumn ASCII table of X Y Z values. For those times when you really want to see the numbers, now you can see them on the screen or save to a file.

21.32.17 Windows

Three options may be set in the windows driver.

Syntax:

```
set terminal windows {<color>} {"<fontname>"} {<fontsize>}
```

<color> is either **color** or **monochrome**. "<fontname>" is the name of a valid Windows font. <fontsize> is the size of the font in points.

Graph-menu The **gnuplot graph** window has the following options on a pop up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Bring to Top when checked brings the graph window to the top after every plot.

Color when checked enables color linestyles. When unchecked it forces monochrome linestyles.

Copy to Clipboard copies a bitmap and a Metafile picture.

Background... sets the window background color.

Choose Font... selects the font used in the graphics window.

Line Styles... allows customization of the line colors and styles.

Print... prints the graphics windows using a Windows printer driver and allows selection of the printer and scaling of the output. The output produced by **Print** is not as good as that from gnuplot's own printer drivers.

Update wgnuplot.ini saves the current window locations, window sizes, text window font, text window font size, graph window font, graph window font size, background color and linestyles to the initialisation file **WGNUPLOT.INI**.

Printing In order of preference, graphs may be be printed in the following ways.

1. Use the gnuplot command **set terminal** to select a printer and **set output** to redirect output to a file.

2. Select the **Print...** command from the **gnuplot graph** window. An extra command **screendump** does this from the text window.

3. If **set output "PRN"** is used, output will go to a temporary file. When you exit from gnuplot or when you change the output with another **set output** command, a dialog box will appear for you to select a printer port. If you choose OK, the output will be printed on the selected port, passing unmodified through the print manager. It is possible to accidently (or deliberately) send printer output meant for one printer to an incompatible printer.

Text-menu The **gnuplot text** window has the following options on a pop up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Copy to Clipboard copies marked text to the clipboard.

Paste copies text from the clipboard as if typed by the user.

Choose Font... selects the font used in the text window.

System Colors when selected makes the text window honor the System Colors set using the Control Panel. When unselected. text is black or blue on a white background.

Update wgnuplot.ini saves the current text window location, text window size, text window font and text window font size to the initialisation file **WGNUPLOT.INI**.

MENU BAR

If the menu file WGNUPLOT.MNU is found in the same directory as WGNUPLOT.EXE, then the menu specified in WGNUPLOT.MNU will be loaded.

Menu commands are:

```
[Menu]       Start a new menu with the name on the following line
[EndMenu]    End current menu.
--           Insert a horizontal menu separator
|            Insert a vertical menu separator
[Button]     Put next macro on a push button instead of a menu.
```

Macros take two lines with the macro name (menu entry) on the first line and the macro on the second line. Leading spaces are ignored.

Macros commands are:

```
[INPUT]      Input string with prompt terminated by [EOS] or {ENTER}
[EOS]        End Of String terminator.  Generates no output.
[OPEN]       Get name of file to open from list box, with title of
             list box terminated by [EOS], followed by default
             filename terminated by [EOS] or {ENTER}
             This uses COMMDLG.DLL from Windows 3.1.
[SAVE]       Get name of file to save.  Similar to [OPEN]
```

Macros character substitutions are:

```
{ENTER}      Carriage Return '\r'
{TAB}        Tab      '\011'
{ESC}        Escape  '\033'
{^A}         '\001'
...
{^_}         '\031'
```

Macros are limited to 256 characters after expansion.

Wgnuplot.ini Windows gnuplot will read some of its options from the **[WGNUPLOT]** section of **WGNUPLOT.INI** in the Windows directory. An example **WGNUPLOT.INI** file is shown below.

```
[WGNUPLOT]
TextOrigin=0 0
TextSize=640 150
TextFont=Terminal,9
GraphOrigin=0 150
GraphSize=640 330
GraphFont=Arial,10
GraphColor=1
GraphToTop=1
GraphBackground=255 255 255
Border=0 0 0 0 0
Axis=192 192 192 2 2
Line1=0 0 255 0 0
Line2=0 255 0 0 1
Line3=255 0 0 0 2
Line4=255 0 255 0 3
Line5=0 0 128 0 4
```

The **GraphFont** entry specifies the font name and size in points. The 5 numbers given in the **Border**, **Axis** and **Line** entries are the **Red** intensity (0-255), **Green** intensity, **Blue** intensity, **Color Linestyle** and Mono Linestyle. Linestyles are 0=SOLID, 1=DASH, 2=DOT, 3=DASHDOT, 4=DASHDOTDOT. In the example **WGNUPLOT.INI** file above, Line 2 is a green solid line in color mode, or a dashed line in monochrome mode. The default line width is 1 pixel. If **Linestyle** is negative it specifies the width of a SOLID line in pixels. Line1 and any linestyle used with the **points** style must be SOLID with unit width.

Windows3.0 Windows 3.1 is preferred, but WGNUPLOT will run under Windows 3.0 with the following restrictions:

1. COMMDLG.DLL and SHELL.DLL (available with Windows 3.1 or Borland C++ 3.1) must be in the windows directory.

2. WGNUPLOT.HLP produced by Borland C++ 3.1 is in Windows 3.1 format. You need to use the WINHELP.EXE supplied with Borland C++ 3.1.

3. It won't run in real mode due to lack of memory.

4. Truetype fonts are not available in the graph window.

5. Drag-drop does not work.

21.33 Tics

By default, tics are drawn inwards from the border on all four sides. The **set tics** command can be used to change the tics to be drawn outwards on the left and bottom borders only. This is useful when doing impulse plots.

Syntax:

```
        set tics {<direction>}
        show tics
```

where <direction> may be **in** or **out**. **set tics** defaults to **in**.

See also the **set xtics**, **set ytics**, and **set ztics** command for more control of tic marks. Using splot, in 3-d plots, one can adjust the relative height of the vertical (Z) axis using **set ticslevel**. The numeric argument provided specifies the location of the bottom of the scale. a zero will put it on the bottom grid and any positive number somewhere along the z axis.

Syntax:

```
        set ticslevel {<level>}
        show tics
```

where <level> is a non negative numeric argument. For example,

```
        set ticslevel 0.5
```

sets the tics level to the default value.

See also the **set view**.

21.34 Time

The optional **set time** places the time and date of the plot either at the top or bottom of the left margin. The exact location is device dependent.

Syntax:

```
        set time {<xoff>}{,<yoff>}
        set notime
        show time
```

Specifying constants <xoff> or <yoff> as optional offsets for the time will move the time <xoff> or <yoff> character screen coordinates. For example,

```
        set time ,-3
```

will change only the y offset of the time, moving the title down by roughly the height of three characters.

21.35 Title

The **set title** command produces a plot title that is centered at the top of the plot. Using the optional x,y screen offsets, the title can be placed anywhere on the plot. **set title** with no parameters clears the title.

Syntax:

```
        set title {"<title-text>"} {<xoff>}{,<yoff>}
        show title
```

Specifying constants <xoff> or <yoff> as optional offsets for the title will move the title <xoff> or <yoff> character screen coordinates. Note these are screen coordinates and not plot coordinates. For example,

```
set title ,-1
```

will change only the y offset of the title, moving the title down by roughly the height of one character.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

21.36 Trange

The **set trange** command sets the parametric range used to compute x and y values when in parametric mode. If not in parametric mode (see **set parametric**) then this range is not used. This command does not affect x/y autoscaling or x/y ranges.

This range may also be specified on the **plot** command line when in parametric mode.

Syntax:

```
set trange [{<tmin> : <tmax>}]
```

where <tmin> and <tmax> terms are constants or expressions.

Both the <tmin> and <tmax> terms are optional. Anything omitted will not be changed, so

```
set trange [:10]
```

changes tmax to 10 without affecting tmin. See also **set urange** and **set parametric**.

21.37 Urange

The **set urange** and **set vrange** commands sets the parametric ranges used to compute x, y, and z values when in splot parametric mode. If not in parametric mode (see **set parametric**) then these ranges are not used. This command does not affect x/y autoscaling or x/y ranges.

This range may also be specified on the **splot** command line when in parametric mode. See **plot** for more information

Syntax:

```
set urange [{<umin> : <umax>}]
```

where <umin> and <umax> terms are constants or expressions.

Both the <umin> and <umax> terms are optional. Anything omitted will not be changed, so

```
set urange [:10]
```

changes umax to 10 without affecting umin. See also **set trange**.

21.38 Variables

The **show variables** command lists all user-defined variables and their values.

Syntax:

```
show variables
```

21.39 View

The set **view** command sets the view point for **splots**. This command controls the way the 3-d coordinates of the plot are mapped into the 2-d screen space. This command provides controls to both rotation and scaling of the plotted data but supports orthographic projections only.

Syntax:

```
set view <rot_x> {,{<rot_z>}{,{<scale>}{,<scale_z>}}}
show view
```

where <rot_x> and <rot_z> control the rotation angles (in degrees) along a virtual 3-d coordinate system aligned with the screen such that the screen horizontal axis is x, screen vertical axis is y, and the axis perpendicular to the screen is z. <rot_x> is bounded to the [0:180] range with a default of 60 degrees, while <rot_z> is bounded to the [0:360] range with a default of 30 degrees. <scale> controls the scaling of the entire **splot**, while <scale_z> scales the z axis only. Both scales default to 1.0.

Examples:

```
set view 60, 30, 1, 1
set view ,,0.5
```

The first sets all the four default values. The second changes only scale, to 0.5.

See also **set ticslevel**.

21.40 Vrange

The set **vrange** command is similar to the **set urange** command. Please see **set urange**.

21.41 Xlabel

The set **xlabel** command sets the x-axis label that is centered along the x axis. Using the optional x,y screen offsets, the label can be placed anywhere on the plot. **set xlabel** with no parameters clears the label.

Syntax:

```
set xlabel {"<label>"} {<xoff>}{,<yoff>}
show xlabel
```

Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
set xlabel -1
```

will change only the x offset of the xlabel, moving the label roughly one character width to the left.

(The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

21.42 Xrange

The set **xrange** command sets the horizontal range that will be displayed. This command turns x axis autoscaling off.

This range may also be specified on the **plot** command line.

Syntax:

```
        set xrange [{<xmin> : <xmax>}]
```

where <xmin> and <xmax> terms are constants or expressions.

Both the <xmin> and <xmax> terms are optional. Anything omitted will not be changed, so

```
        set xrange [:10]
```

changes xmax to 10 without affecting xmin.

21.43 Xtics

Fine control of the x axis tic marks is possible with the **set xtics** command. The x-axis tic marks may be turned off with the **set noxtics** command. They may be turned on (the default state) with **set xtics**.

Syntax:

```
         set xtics { {<start>, <incr>{, <end>}} |
                    {({"<label>"} <pos> {, {"<label>"} <pos>}...)} }
         set noxtics
         show xtics
```

The <start>, <incr>. <end> form specifies that a series of tics will be plotted on the x axis between the x values <start> and <end> with an increment of <incr>. If <end> is not given it is assumed to be infinity. The increment may be negative. For example,

```
          set xtics 0,.5,10
```

makes tics at 0, 0.5, 1, 1.5, ..., 9.5, 10.

The ("<label>" <pos>, ...) form allows arbitrary tic positions or non-numeric tic labels. A set of tics are a set of positions. each with its own optional label. Note that the label is a string enclosed by quotes, and may be a constant string, such as "hello", or contain formatting information for the tic number (which is the same as the position), such as "%3f clients". See set format for more information about this case. The label may even be empty. Examples:

```
         set xtics ("low" 0, "medium" 50, "high" 100)
         set xtics (1,2,4,8,16,32,64,128,256,512,1024)
         set xtics ("bottom" 0, "" 10, "top" 20)
```

Tics will only be plotted when in range.

The **set ytics** and **set noytics** commands work identically. See also the **set format** command.

21.44 Xdtics

The **set xdtics** commands converts the x axis tic marks to days of the week where 0=Sun and 6=Sat. Overflows are converted modulo 7 to dates.

Examples:

```
        set xdtics
```

Sets x axis tics in days.

The **set ydtics set zdtics** and **set noydtics set nozdtics** commands work identically. See also the **set format** command.

21.45 Xmtics

The set **xmtics** commands converts the x axis tic marks to months of the years where 1=Jan and 12=Dec. Overflows are converted modulo 12 to months.

Examples:

```
      set xmtics
```

Sets x axis tics into months.

The set **ymtics set zmtics** and **set noymtics set nozmtics** commands work identically. See also the set **format** command.

21.46 Xzeroaxis

set xzeroaxis draws the x-axis. By default, this option is on. **set noxzeroaxis** causes GNUPLOT to omit the x-axis.

Syntax:

```
       set xzeroaxis
       set noxzeroaxis
       show xzeroaxis
```

21.47 Ylabel

The set **ylabel** command sets the y-axis label. The position of this label depends on the terminal, and can be one of the following three positions (the position can be adjusted with optional parameters).

1. Horizontal text flushed left at the top left of the plot. Terminals that cannot rotate text will probably use this method.

2. Vertical text centered vertically at the left of the plot. Terminals that can rotate text will probably use this method.

3. Horizontal text centered vertically at the left of the plot. The EEPIC, LaTeX and TPIC drivers use this method. The user must insert line breaks using \\ to prevent the ylabel from overwriting the plot. To produce a vertical row of characters, add \\ between every printing character (but this is ugly).

Syntax:

```
        set ylabel {"<label>"} {<xoff>}{,<yoff>}
        show ylabel
```

With no parameters, the label is cleared. Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
        set ylabel -1
```

will change only the x offset of the ylabel, moving the label roughly one character width left of its default position. This is especially useful with the LaTeX driver.

(The EEPIC. Imagen. LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

21.48 Yrange

The set **yrange** command sets the vertical range that will be displayed. This command turns y axis autoscaling off.

This range may also be specified on the **plot** command line.

Syntax:

```
        set yrange [{<ymin> : <ymax>}]
```

where <ymin> and <ymax> terms are constants or expressions.

Both the <ymin> and <ymax> terms are optional. Anything omitted will not be changed, so

```
        set yrange [:10]
```

changes ymax to 10 without affecting ymin.

21.49 Ytics

The **set ytics** and **set noytics** commands are similar to the **set xtics** and **set noxtics** commands. Please see **set xtics**.

21.50 Ydtics

The **set ydtics** and **set noydtics** commands are similar to the **set xdtics** and **set noxdtics** commands. Please see **set xdtics**.

21.51 Ymtics

The **set ymtics** and **set noymtics** commands are similar to the **set xmtics** and **set noxmtics** commands. Please see **set xmtics**.

21.52 Yzeroaxis

set yzeroaxis draws the y-axis. By default, this option is on. **set noyzeroaxis** causes GNUPLOT to omit the y-axis.

Syntax:

```
        set yzeroaxis
        set noyzeroaxis
        show yzeroaxis
```

21.53 Zero

The **zero** value is the default threshold for values approaching 0.0. GNUPLOT will not plot a point if its imaginary part is greater in magnitude than the **zero** threshold. Axis ranges cannot be less than zero. The default **zero** value is 1e-8. This can be changed with the **set zero** command.

Syntax:

```
        set zero <expression>
        show zero
```

21.54 Zeroaxis

set zeroaxis draws the x-axis and y-axis. By default, this option is on. **set nozeroaxis** causes GNUPLOT to omit the axes, and is equivalent to **set noxzeroaxis; set noyzeroaxis.**

Syntax:

```
        set zeroaxis
        set nozeroaxis
        show zeroaxis
```

See **set xzeroaxis** and **set yzeroaxis.**

21.55 Zlabel

The **set zlabel** command sets the z-axis label that is centered along the z axis. Using the optional x,y screen offsets, the label can be placed anywhere on the plot. **set zlabel** with no parameters clears the label.

Syntax:

```
        set zlabel {"<label>"} {<xoff>}{,<yoff>}
        show zlabel
```

Specifying constants <xoff> or <yoff> as optional offsets for the label will move the label <xoff> or <yoff> character screen coordinates. For example,

```
        set zlabel ,1
```

will change only the y offset of the zlabel, moving the label roughly one character height up.

The zlabel will be drawn whenever surfaces or contours are plotted, in the space above the grid level.

(The EEPIC. Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.)

21.56 Zrange

The **set zrange** command sets the vertical range that will be displayed. This command turns z axis autoscaling off. The zrange is used only by **splot** and is ignored by **plot.**

This range may also be specified on the **splot** command line.

Syntax:

```
        set zrange [{<zmin> : <zmax>}]
```

where <zmin> and <zmax> terms are constants or expressions.

Both the <zmin> and <zmax> terms are optional. Anything omitted will not be changed, so

```
        set zrange [2:]
```

changes zmin to 2 without affecting zmax.

21.57 Ztics

The **set ztics** and **set noztics** commands are similar to the **set xtics** and **set noxtics** commands. Please see **set xtics.**

21.58 Zdtics

The set **zdtics** and set **nozdtics** commands are similar to the set **xdtics** and set **noxdtics** commands. Please see set **xdtics**.

21.59 Zmtics

The set **zmtics** and set **nozmtics** commands are similar to the set **xmtics** and set **noxmtics** commands. Please see set **xmtics**.

22 Shell

The **shell** command spawns an interactive shell. To return to GNUPLOT, type **logout** if using VMS, **exit** or the END-OF-FILE character if using Unix, **endcli** if using AmigaDOS, or **exit** if using MS-DOS or OS/2.

A single shell command may be spawned by preceding it with the ! character ($ if using VMS) at the beginning of a command line. Control will return immediately to GNUPLOT after this command is executed. For example, in VMS, AmigaDOS, MS-DOS or OS/2,

```
! dir
```

prints a directory listing and then returns to GNUPLOT.

On an Atari, the **!** command first checks whether a shell is already loaded and uses it, if available. This is practical if GNUPLOT is run from **gulam**, for example.

23 Splot

Three-dimensional surface and contour plotting is available in GNUPLOT with the **splot** command. See the **plot** command for features common to the **plot** command.

See also set **contour**, set **cntrparam**, and set **surface**.

23.1 Binary Data

Gnuplot will dynamically determine if a datafile is ASCII or binary. ASCII data files are discussed in the **plot** section. For three dimensions, single precision floats are stored as follows:

```
<ncols> <x0> <x1> <x2> ...
<y0> <z0,0> <z0,1> <z0,2> ...
<y1> <z1,0> <z1,1> <z1,2> ...
```

which is converted into triplet:

```
<x0> <y0> <z0,0>
<x0> <y1> <z0,1>
<x0> <y2> <z0,2>

<x1> <y0> <z1,0>
<x1> <y1> <z1,1>
<x1> <y2> <z1,2>
```

These triplets are then converted into gnuplot iso_curves and then uses gnuplot to do the rest of the plotting.

A collection of matrix and vector manipulation routines (in C) are provided in **gnubin.c**. The routine to write binary data is

```
 int fwrite_matrix(file,m,nrl,nrl,ncl,nch,row_title,column_title)
```

An example of using these routines is provided in the file **bf_test.c**. The corresponding demo file is **demo/binary.dem**.

24 Start-up

When GNUPLOT is run, it looks for an initialization file to load. This file is called **.gnuplot** on Unix and AmigaDOS systems, and **GNUPLOT.INI** on other systems. If this file is not found in the current directory, the program will look for it in the home directory (under AmigaDOS, AtariTOS, MS-DOS and OS/2, the environment variable GNUPLOT should contain the name of this directory). Note: if NOCWDRC is defined during the installation, GNUPLOT will not read from the current directory.

If this file is found, GNUPLOT executes the commands in this file. This is most useful for setting the terminal type and defining any functions or variables that are used often.

25 Substitution

Command-line substitution is specified by a system command enclosed in backquotes. This command is spawned and the output it produces replaces the name of the command (and backquotes) on the command line.

Newlines in the output produced by the spawned command are replaced with blanks.

Command-line substitution can be used anywhere on the GNUPLOT command line.

Example:

This will run the program leastsq and replace leastsq (including backquotes) on the command line with its output:

```
        f(x) = 'leastsq'
```

or, in VMS

```
        f(x) = 'run leastsq'
```

26 User-defined

New user-defined variables and functions of one through five variables may be declared and used anywhere.

User-defined function syntax:

```
        <function-name> ( <dummy1> {,<dummy2> {, ...} } ) = <expression>
```

where <expression> is defined in terms of <dummy1> through <dummy5>.

User-defined variable syntax:

```
        <variable-name> = <constant-expression>
```

Examples:

```
        w = 2
        q = floor(tan(pi/2 - 0.1))
        f(x) = sin(w*x)
        sinc(x) = sin(pi*x)/(pi*x)
        delta(t) = (t == 0)
        ramp(t) = (t > 0) ? t : 0
        min(a,b) = (a < b) ? a : b
        comb(n,k) = n!/(k!*(n-k)!)
        len3d(x,y,z) = sqrt(x*x+y*y+z*z)
```

Note that the variable **pi** is already defined.

See **show functions** and **show variables**.

27 Bugs

The bessel functions do not work for complex arguments.

The gamma function does not work for complex arguments.

There is a bug in the stdio library for old Sun operating systems (SunOS Sys4-3.2). The "%g" format for 'printf' sometimes incorrectly prints numbers (e.g., 200000.0 as "2"). Thus, tic mark labels may be incorrect on a Sun4 version of GNUPLOT. A work-around is to rescale the data or use the **set format** command to change the tic mark format to "%7.0f" or some other appropriate format. This appears to have been fixed in SunOS 4.0.

Another bug: On a Sun3 under SunOS 4.0, and on Sun4's under Sys4-3.2 and SunOS 4.0, the 'sscanf' routine incorrectly parses "00 12" with the format "%f %f" and reads 0 and 0 instead of 0 and 12. This affects data input. If the data file contains x coordinates that are zero but are specified like '00', '000', etc, then you will read the wrong y values. Check any data files or upgrade the SunOS. It appears to have been fixed in SunOS 4.1.1.

Microsoft C 5.1 has a nasty bug associated with the %g format for printf. When any of the formats "%.2g", "%.1g", "%.0g", "%.g" are used, printf will incorrectly print numbers in the range 1e-4 to 1e-1. Numbers that should be printed in the %e format are incorrectly printed in the %f format, with the wrong number of zeros after the decimal point.

To work around this problem, use the %e or %f formats explicitly.

GNUPLOT, when compiled with Microsoft C, did not work correctly on two VGA displays that were tested. The CGA, EGA and VGA drivers should probably be rewritten to use the Microsoft C graphics library. GNUPLOT compiled with Borland C++ uses the Turbo C graphics drivers and does work correctly with VGA displays.

VAX/VMS 4.7 C compiler release 2.4 also has a poorly implemented %g format for printf. The numbers are printed numerically correct, but may not be in the requested format. The K&R second edition says that for the %g format, %e is used if the exponent is less than -4 or greater than or equal to the precision. The VAX uses %e format if the exponent is less than -1. The VAX appears to take no notice of the precision when deciding whether to use %e or %f for numbers less than 1. To work around this problem, use the %e or %f formats explicitly. From the VAX C 2.4 release notes: e,E,f,F,g,G Result will always contain a decimal point. For g and G, trailing zeros will not be removed from the result.

VAX/VMS 5.2 C compiler release 3.0 has a slightly better implemented %g format than release 2.4, but not much. Trailing decimal points are now removed, but trailing zeros are still not removed from %g numbers in exponential format.

ULTRIX X11R3 has a bug that causes the X11 driver to display "every other" plot. The bug seems to be fixed in DEC's release of X11R4 so newer releases of ULTRIX don't seem to have the problem. Solutions for older sites include upgrading the X11 libraries (from DEC or direct from MIT) or defining ULTRIX_KLUDGE when compiling the x11.trm file. Note that the kludge is not an ideal fix, however.

The constant HUGE was incorrectly defined in the NeXT OS 2.0 operating system. HUGE should be set to 1e38 in plot.h. This error has been corrected in the 2.1 version of NeXT OS.

Some older models of HP plotters do not have a page eject command 'PG'. The current HPGL driver uses this command in HPGL_reset. This may need to be removed for these plotters. The current PCL5 driver uses HPGL/2 for text as well as graphics. This should be modified to use scalable PCL fonts.

On the Atari version, it is not possible to send output directly to the printer (using **/dev/lp** as output file), since CRs are added to LFs in binary output. As a workaround write the output to a file and copy it to the printer afterwards using a shell command.

Please report any bugs to bug-gnuplot@dartmouth.edu.