ORNL/M--828

DE90 005815

# Using the K-25 C&TD Common File System: A Guide to CFSI

C&TD User Services

Computing and Telecommunications Division
at Oak Ridge National Laboratory
Post Office Box 2008
Oak Ridge, Tennessee 37831

Date Published - December 1989

MARTIN MARIETTA ENERGY SYSTEMS, INC.
operating the
Oak Ridge National Laboratory Oak Ridge Y-12 Plant
Oak Ridge Gaseous Diffusion Plant Paducah Gaseous Diffusion Plant
under contract DE-AC05-84OR21400
for the
U.S. Department of Energy

# Table of Contents

## Abstract

A CFS (Common File System) is a large, centralized file management and storage facility based on software developed at Los Alamos National Laboratory. This manual is a guide to use of the CFS available to users of the Cray UNICOS system at Martin Marietta Energy Systems, Inc. in Oak Ridge, Tennessee.

# Preface

The subjects of this manual are

- the K-25 Common File System (CFS), which is available to users of the UNICOS Cray for file storage, and
- the CFS Interface (`cfsi`) program used to access the K-25 CFS.*

This manual is intended to serve as a primer for new CFS users and also as a quick reference for more experienced users. Additional CFS features are described in the *Common File System CFS Interface Reference*, which is available in the Programming Assistance offices.

For help with the K-25 CFS, contact Programming Assistance at your site.

| | |
|---|---|
| K-25 | 4-8837 |
| X-10 | 4-5321 |
| Y-12 | 6-5908 |

This manual is organized as follows:

- Chapters 1 through 3 explain the basic concepts you need to understand before beginning to use CFS.
- Chapter 4 contains a tutorial to get you started with basic operations.
- Chapter 5 explains CFS charges and how they are assessed.
- Chapters 6 through 8 cover additional CFS features, including file protection, that you may want to take advantage of.
- Chapters 9 through 11 provide complete descriptions of individual commands, parameters, and keywords. The alphabetical listings in each of these chapters is preceded by a short introductory section that explains the format and conventions used and provides reminders of where to go for related information.

* A `cfsi` program is not available on the Cray under CTSS; to access CFS from CTSS, use **mass**.

- The appendixes provide supplementary information about using the K-25 CFS.

Several conventions apply throughout the remainder of this manual.

- For ease of reading, and in line with common usage, the articles *a* and *the* are generally omitted where they would ordinarily be expected to precede the terms *CFS* and `cfsi`.
- Some information is boxed for emphasis.
- Italics are used in the text to introduce new terms, for emphasis, and to clarify the use of certain terms.
- The term *userid* refers to a CFS user identifier. On the K-25 CFS, this identifier is the same as a user's Cray User Number. As explained later in this manual, that number is reserved for its owner to use as a userid root name; it is also the number that must be used to identify the user in validation entries.

  Your Cray User Number is derived from the three-character UID you use to log on to the Cray, with each character of your UID being replaced with the appropriate pair of digits from the chart below.

  | | | | | | |
  |---|---|---|---|---|---|
  | A=01 | B=02 | C=03 | D=04 | E=05 | F=06 |
  | G=07 | H=08 | I=09 | J=10 | K=11 | L=12 |
  | M=13 | N=14 | O=15 | P=16 | Q=17 | R=18 |
  | S=19 | T=20 | U=21 | V=22 | W=23 | X=24 |
  | Y=25 | Z=26 | 0=30 | 1=31 | 2=32 | 3=33 |
  | 4=34 | 5=35 | 6=36 | 7=37 | 8=38 | 9=39 |

  For example, if your UID is ABC, your Cray User Number is 010203. Your Cray User Number is displayed as your *Z number* when you invoke `cfsi`.
- In the examples, lowercase monospaced entries indicate *literal* values or names that must be entered as shown. Lowercase entries are *symbolic* values that you are expected to replace with your own, user-specific information. For example, if you are instructed to enter

  **get** *filename*

  you must enter **get**, followed by the name of the specific file you want to retrieve.

  When the symbolic value *userid* is indicated, unless otherwise instructed, substitute your Cray User Number. Examples generally use the arbitrarily selected identifiers 010203, 040506, and 070809 to refer to other users.

- For simplicity, in examples that copy a file to or from a UNICOS directory, it is assumed that you invoked `cfsi` from that directory. You can specify other directories as the file source or destination; see Chapter 10, Section 10.4 and Appendix C for instructions.

As explained in Chapter 2, you use CFS by entering requests through the `cfsi` program that runs on the UNICOS Cray. When using CFS interactively, you enter the requests at the `cfsi ?` prompt and indicate the end of each request by pressing <RETURN>. In this manual, `cfsi` prompts and <RETURN>s are omitted from examples.

| |
|---|
| This manual was tested under CFS Version 56c. |

# 1. A Common File System (CFS) Overview

A Common File System (CFS) is a large, centralized file* management and storage program that runs on a dedicated computer. One or more computing systems that are used to create and manipulate files may be connected to a CFS. Such systems are known as *worker systems*. The Cray is the only worker system attached to the K-25 CFS. If you are a Cray user, you can transfer files quickly and easily to CFS for storage. Retrieval of files for use is just as easy. You can also

- copy, move, replace, or delete stored files;
- organize stored files into simple or complex tree structures;
- share your stored files with other users who have access to the K-25 CFS; and
- access other users' files when the required permission has been granted.

By default, you are the only user who can access your CFS files.

You communicate with CFS through a CFS Interface (`cfsi`) program on the UNICOS Cray. Using `cfsi`, you make *requests* (e.g., to retrieve a file), which are then executed by CFS. CFS returns a *response* for each request.

This manual presents `cfsi` from the perspective of interactive use. However, use of `cfsi` in the background and in batch jobs is basically the same; for information, see Chapter 2, Section 2.5.

CFS handles only complete files; access to partial files is not permitted. The maximum allowable file size is currently 190 million bytes. If your file exceeds that limit, use `dd` or `split` on UNICOS to divide the file into storable sections and `cat` to reassemble the parts after retrieval from CFS.

* A file on a CFS is any string of bits with a unique name. The bits can represent text, input data, source code, output data, etc.

CFS stores files either online on high-speed disk or offline on tape cartridge. In each case, the device is selected by the CFS program to provide the fastest response time to the most frequently used files. A new file is stored initially on disk. A file migration program within CFS then moves files between disk and tape based on the frequency of access of the file. Thus, when a file is stored on disk, it is automatically moved to tape if it is not accessed during a system-set time period. The file remains on tape unless

- it is deleted and stored again by the user, or
- it is automatically moved to disk when the frequency of access reaches a system-set value, which is based on the amount and type of available storage.

The file migration program also deletes files with expired release dates.

The response time for retrieving files from CFS is generally on the order of a few seconds from disk; from tape, response time is usually on the order of two to four minutes, but can be longer in certain cases (e.g., all tape drives are already in use).

CFS handles files as strings of bits (the basic unit of computer information) and does not attempt to interpret or change the format of any file. Thus, if you stored files from CTSS using **mass** and want to use those files on UNICOS, you must convert them from CTSS to UNICOS format. Appendix B contains conversion instructions for text files. Conversion of binary files is usually not possible.

No special validation other than what is required to log on to UNICOS is needed to use the K-25 CFS.

Charges for your use of CFS are assessed to the C&TD computer charge number(s) you provide. The rate for use of CFS disk and tape space is considerably lower than the rate for use of disk space on UNICOS. For more information, see Chapter 5 and **man charges** on UNICOS.

Because the storage devices used are extremely reliable, there are no automatic system backups of files stored on CFS. If your files are in the security category *mission essential*, you should provide backup as described in Chapter 11, Sections 11.6, 11.12, and 11.18.

# 2. Working with CFS

This chapter is an introduction to the use of CFS requests and to the responses CFS gives to these requests. Many of the operations you can perform using requests are mentioned. Actual how-tos, including instructions for invoking and exiting the CFS Interface program `cfsi`, are provided in tutorial form in Chapter 4.

This manual presents `cfsi` from the perspective of interactive use. However, use of `cfsi` in the background and in batch jobs is basically the same; for information, see Section 2.5.

## 2.1 The Mechanics of Requests and Responses

To use CFS interactively, you

- invoke `cfsi`,
- enter CFS requests at the `cfsi` prompt (a `?`),
- check the response that is returned to each request, then
- exit `cfsi` and return to the UNICOS system prompt.

As an alternative, you can include the `cfsi` call and multiple requests in one execution line followed by a `<RETURN>`. This is known as using `cfsi` in *single-line execution mode*. All requests on the line are executed and `cfsi` returns you to the UNICOS system prompt. For more information, see Section 2.4.

When you enter a request, `cfsi` sends it to the CFS processor and waits for a response. After a given interval of time has passed with no response, `cfsi` sends a status query to CFS (see Chapter 9, Section 9.18) and returns an appropriate response to you. `cfsi` will repeat this procedure periodically for about an hour or until CFS executes your request. If you are an interactive user and want to interrupt this query cycle, enter `<CTRL-c>`.

If the `cfsi` status query determines that CFS is down or that the network connecting the Cray to CFS is down, `cfsi` will resend your request periodically for about an hour or until CFS receives it. If you are an interactive user, you can enter `<CTRL-c>` to interrupt a retry cycle.

## 2.2 Requests

The basic format for all CFS requests is the *command* (or *request*) *line* shown below

*command parameter keyword=value <comment>*

where

- *command*

  describes the desired action, such as storing or retrieving a file.

- *parameter*

  specifies the object on which the command action is to be applied (e.g., a file).

- *keyword=value*

  specifies additional information on the request.

- *<comment>*

  is a string of text information that documents a command. The angle brackets (<>) are required.

Each request must begin with a command; the other elements of a request may follow in any order. To end a request line, enter `<RETURN>`.

All input to **`cfsi`**, including file names, must be in lowercase. An enhancement is being developed that will permit the use of mixed-case and uppercase file names. Meanwhile, to make file manipulation easy, use only lowercase characters when naming your UNICOS files or refer to the handout *Converting to UNICOS* for special procedures that will allow you to preserve case in file names on CFS. The handout is available from Programming Assistance.

Sections 2.2.1 through 2.2.4 provide more details about each component of a CFS request. Section 2.2.5 lists some information pertaining to the use of CFS requests.

### 2.2.1 Commands

A command is a literal name that specifies an action to be performed. The first word of each CFS request must be a command; each request may contain only one command. Most commands can be abbreviated. Each command is described in detail in Chapter 9.

Table 2.1 provides a summary of CFS commands, by function. Table 2.2 lists the shortest acceptable abbreviation for each of these commands. Of necessity, some terms that have not yet been introduced appear in the command summaries. These terms are explained in Chapters 3 through 8.

Table 2.1. Descriptions of CFS commands grouped according to general function

| | |
|---|---|
| **Commands for manipulating roots and subdirectories** | |
| `add` | Adds a subdirectory to an existing parent directory. |
| `create`[a] | Creates a root directory. |
| `move` | Moves an existing CFS node and its subtrees, if any, to another node; can create a new root node. |
| `remove` | Removes a subdirectory or root directory. |
| **Commands for manipulating files** | |
| `copy` | Copies an existing CFS file into a new CFS file. |
| `delete`[a] | Deletes a file from CFS. |
| `get`[a] | Copies a file from CFS to UNICOS. |
| `replace` | Replaces an existing CFS file with a file from UNICOS. |
| `save`[a] | Saves a UNICOS file as a new file on CFS. |
| `store` | Executes a `save` if the file name does not exist; a `replace` if the file name does exist. |
| **Commands for changing administrative information in nodes** | |
| `modify` | Modifies the content of a directory node or file descriptor. Adds or changes user validation entries; estimate of file activity; file or directory name; charging information, file release date, etc. |
| **Commands for changing keyword values** | |
| `set` | Sets values for keywords in the active keyset. Values remain in effect until redefined by another `set` or until an `adopt` replaces all keyword values or the session is ended. |
| **Commands for working with saved keysets** | |
| `adopt` | Re-establishes a saved keyset as the active keyset. |
| `free` | Deletes a saved keyset. |
| `keep` | Stores an active keyset as a saved keyset for future use. |
| `show` | Displays names and/or contents of saved keysets. |
| **Miscellaneous commands** | |
| `end`[a] | Terminates `cfsi`. |
| `list`[a] | Returns information about a directory or file; may include user validation entries, immediate descendants, etc. |
| `status`[a] | Outputs the status of CFS. |

[a] This command is one of the small set needed by most users who just want to store a few files on CFS using their userid root.

**Table 2.2. CFS commands and their shortest permissible abbreviations**

| Command | Abbreviation |
|---|---|
| `add` | `a` |
| `adopt` | `ado` |
| `copy` | `c` |
| `create` | `cr` |
| `delete` | `del` |
| `end` | `en` |
| `free` | `fr` |
| `get` | `g` |
| `keep` | `k` |
| `list` | `l` |
| `modify` | `mod` |
| `move` | `mov` |
| `remove` | `rem` |
| `replace` | `r` |
| `save` | `s` |
| `set` | `se` |
| `show` | `sh` |
| `status` | `sta` |
| `store` | `st` |

## 2.2.2 Parameters

A parameter specifies the object upon which the associated command is to be performed. Multiple parameters are permitted for most commands, as shown for **save** in the following example:

**save** *filename1 filename2*

Each parameter must be separated from the next by a space.

Four types of parameters are accepted by CFS. These are listed below and discussed in Chapter 10 and the additional references cited.

- **Root:** the name of a CFS root directory node (may be a userid or other name). See Chapter 3, Section 3.1.
- **Path:** a specification for a CFS file or directory node. See Chapter 3, Section 3.2.
- **Workerfile name:** (also known as *local file name*) the name of a UNICOS working directory file or the path (relative to the working directory) for a UNICOS file.
- **Keyset:** the name of a keyset. See Section 2.2.3 and Chapter 7.

For some commands, a null (i.e., a blank) and/or a minus sign (-) can be used to represent a parameter. For more information, see Chapter 10 and individual command descriptions in Chapter 9.

Each command has its own parameter requirements (see Table 2.3 and the individual command descriptions in Chapter 9). For instance, the commands **adopt**, **keep**, and **free**, which work with saved keysets, require a keyset parameter.

The basic set of characters that can be used in request parameters is composed of the alphanumerics (all letters and the digits 0 through 9) and the following characters:

**$ % * + - . _**

Table 2.3. Parameter requirements for each CFS command

| Command | Parameter Requirement[a] |
|---|---|
| `add` | *path ...* |
| `adopt` | *keyset* |
| `copy` | *path1* `to` *path2 ...* |
| `create` | *root ...* |
| `delete` | *path ...* |
| `end` | |
| `free` | *keyset ...* |
| `get` | *path ...* or *workerfile:path ...* |
| `keep` | *keyset* |
| `list` | *path ...* |
| `modify` | *path ...* |
| `move` | *path1* `to` *path2 ...* |
| `remove` | *path ...* |
| `replace` | *path ...* or *workerfile:path ...* |
| `save` | *path ...* or *workerfile:path ...* |
| `set` | |
| `show` | *keyset ...* |
| `status` | |
| `store` | *path ...* or *workerfile:path ...* |

[a] A series of three elipses (...) indicates that multiples of the associated parameter are allowed. See Chapter 3, Sections 3.1.1 and 3.2, for explanations of *root* and *path*; Chapter 10 for an explanation of *workerfile*; and Chapter 7 for a discussion of *keysets*.

## 2.2.3 Keywords

Keywords are literal names to which values may be assigned. They are used to define node attributes or conditions for a request or to exercise display options.

- Most keywords are used to define node attributes. In other words, the value assigned to the keyword becomes the content of the corresponding field in a directory or file descriptor node when a request using the keyword is executed. (Nodes are discussed in Chapter 3; node fields/attributes are introduced in Chapter 3, Section 3.1 and discussed further in Appendix A.) Thus, you can use **charge=000***nnnnn* with **create** to define the *charge* field for a root node, where *nnnnn* is a C&TD computer charge number. Charges associated with files stored under that root will then be made to charge number *nnnnn*.
- Keywords used to define conditions for a request are **dir***n***=**, **pw***n***=**, and **kpw=**. For example, if a password is required for access to a file, you must use **pw***n***=** to satisfy that condition for access. The use of **dir***n***=** and **pw***n***=** is explained in detail in Chapter 6.
- Keywords used to exercise display options are **lo=**, **lpw=**, and **so=**. For example, you have the option of whether or not to display any user-defined passwords when listing certain information on your terminal; you must use **lpw=on** to display passwords.

As explained later in this section, not all keywords can be used with all commands.

The keywords used in this manual are described in Table 2.4 and, in more detail, in Chapter 11.

Keywords can be used in single-value form as shown below

*keyword=value*

or, for some keywords, in multi-value form with the following format (**lo=** is an exception; see Chapter 11, Section 11.9):

*keyword=(value1 value2 ...)*

**Table 2.4. Some frequently-used keywords and their default values**

| Keyword | Function | Default Value |
|---|---|---|
| **aval=** | Add a user validation entry to an existing node; make the initial change in the master user validation entry for a non-root node | null |
| **charge=** | Define a charge number for storage and access charges | default number |
| **cval=** | Change a user validation entry or a previously changed master user validation entry for an existing node; change the master user validation entry for a root | null |
| **dir*n*=** | Where *n* is 0, define a path for the default working directory | userid root |
| | Where *n* is one of the digits 1-9, define a working directory path | null |
| **dval=** | Delete a user validation entry from an existing node | null |
| **grp=** | Define the storage group for a file | CFS chooses |
| **info=** | Insert descriptive text in a directory node | null |
| **kpw=** | Set a password for a saved keyset | null |
| **lo=** | Use with **list** to select various types of information depending on value option specified and accompanying parameter | **g** for a file<br>**d** for a directory |
| **lpw=** | State whether passwords will be displayed | **off** (no) |
| **ncharge=** | Define a new charge number for use with a root node | null |
| **ngrp=** | Set a storage group for a copied file | null |
| **ninfo=** | Redefine the descriptive text field for a node | null |
| **nname=** | Select a new name for a node | null |
| **nrel=** | Set a new file release date | null |
| **nuse=** | Specify a new estimate of file activity | null |
| **pw*n*=** | Set a password for use with the working directory identified by **dir*n***, where *n* is one of the digits 0 through 9 | null |
| **setgrp=** | Lock a file into the current storage group | null |
| **so=** | Select information to be output with **show** | **names** (i.e., **n**) |

In the multi-value form, values must be separated from each other by at least one space; spaces are permitted before and after the =; and the parentheses are required. The values may be literal or symbolic, depending on the associated keyword. For example, the **pw*n*=** keyword requires a symbolic value (a user-defined password), while a value specified in association with **lo=** must literally be one or more of the six letters **a**, **d**, **g**, **i**, **s**, or **u**, each of which has a specific meaning. Selected value options for each keyword are explained in Chapter 11.

A complete set of keywords and all their values is called a *keyset*. Keysets are discussed in Chapter 7. When you invoke **cfsi**, an *active keyset* is created with each keyword set to its default value (see Table 2.4). Most of the default values are nulls.

Each command has certain keywords that are associated with it.* When you enter a request, CFS automatically gets the values for any of these keywords not defined in the request from your active keyset.

You can change the value associated with any keyword in your active keyset for the remainder of your session or until changed again by using the **set** command as follows:

**set** *keyword=value*

However, in this manual, this request is used only with **dir*n*=**, **pw*n*=**, and **lpw=**. The values of other keywords are changed only for the duration of a request by specifying the keyword and value in a request with a command other than **set** as in the following example:

**list** */userid* **lo=u**

After execution of the request, the value of **lo=** in the active keyset returns to the default (**d**; see Table 2.4).

You can replace your active keyset with a keyset you have previously saved. See Chapter 7 for information about keysets.

* Only the options shown in Table 2.5 are covered in this manual. For a complete list, see Table 7.2.

Use standalone keywords (keywords with no = sign or value) with the **set** command to view the current value of a keyword in your active keyset. For example,

```
set charge
```

causes the system to respond with the current value of the keyword **charge=**. To display your entire active keyset, enter **set**.

**Table 2.5. Commands and their keyword options[a]**

| Command | Keywords | | | |
|---|---|---|---|---|
| **add** | dir*n*= | info= | pw*n*= | |
| **adopt** | kpw= | | | |
| **copy** | dir*n*= | ngrp= | pw*n*= | |
| **create** | charge= | info= | pw*n*= | |
| **delete** | dir*n*= | pw*n*= | | |
| **end** | | | | |
| **free** | kpw= | | | |
| **get** | dir*n*= | pw*n*= | | |
| **keep** | kpw= | | | |
| **list** | dir*n*= | lo= | lpw= | pw*n*= |
| **modify** | aval= | cval= | dir*n*= | dval= |
| | ncharge= | ninfo= | nname= | nrel= |
| | nuse= | pw*n*= | setgrp= | |
| **move** | dir*n*= | pw*n*= | | |
| **remove** | dir*n*= | pw*n*= | | |
| **replace** | dir*n*= | pw*n*= | | |
| **save** | dir*n*= | grp= | pw*n*= | |
| **set**[b] | dir*n*= | lpw= | pw*n*= | |
| **show** | kpw= | lpw= | so= | |
| **status** | | | | |
| **store** | dir*n*= | grp= | pw*n*= | |

[a] Only the options, if any, covered in this manual are listed. For a complete list, see Table 7.2.
[b] Any keyword may be used in standalone form with **set**.

## 2.2.4 Comments

Use the comment field to include text information in a request. The comment must be enclosed in angle brackets (<>). CFS does not process the contents of the comment field, nor does it echo the contents back to you in the request response. Therefore, the main use of the comment field is to document the purpose of CFS requests that are included in batch jobs.

Only one comment may be included in any one command line. A comment may not be embedded in a parameter or in a keyword and its associated values.

## 2.2.5 Information pertaining to the use of CFS requests

The following statements apply when using CFS requests:

- Requests are sent from the CFS Interface on your worker system to a CFS processor.
- CFS returns a response for each request it attempts to execute.
- Each request must begin with a command.
- Parameters, keywords, and a comment can be in any order following a command. For example, a request entered in either of the formats shown below means the same thing to CFS:

  *command parameter <comment> keyword=value*

  or

  *command <comment> keyword=value parameter*

- At least one space must separate the command word and each individual parameter, comment, and keyword.
- A request containing a syntax error in a parameter or keyword is not executed.

- If a keyword is repeated within a request, the last value input is used. For example, if you enter the following request, where *xxxx* and *yyyy* are valid C&TD computer charge numbers,

  ```
  create charge=000xxxx charge=000yyyy
  ```

  the charge number for your new userid root will be *yyyy*.

- If a command is associated in a request with multiple parameters, CFS treats each command/parameter combination as a separate request. Thus,

  *command parameter1 parameter2 keyword=value*

  is understood by CFS as the two requests

  *command parameter1 keyword=value*

  and

  *command parameter2 keyword=value*

  CFS attempts to execute each request, even if one causes an error. A separate response is generated for each request.

- Multiple requests may be entered on the same request line if they are separated by commas as shown in the example below.

  ```
  create, set lpw=on , status
  ```

  Spaces may be placed on either side of commas, but are not required.

- If there is an error in one request of a multiple-request line, no attempt is made to execute any following requests. Thus, if the = sign had been omitted from the example above, an error message would have been returned for the **set** command and the **status** request would have been ignored.

- Normally, <RETURN> signals the end of a request line. However, if you have not finished your request line and want to continue it on another terminal line, you can enter a backslash (\) or a pound sign (#) at the end of the current line,* press <RETURN>, wait for the `continue-` prompt, and continue typing. For example, CFS views the following three lines on a terminal screen as a single request (prompts and <RETURN>s are shown for clarity):

  ```
  ? command parameter1 parameter2 \<RETURN>
  continue-keyword1=value \<RETURN>
  continue-keyword2=value<RETURN>
  ```

  The continuation character is not replaced by a space, so be sure to enter a space before the continuation character or at the beginning of the continuation line, if needed to separate what you have entered at the end of one line and the beginning of the next. Request lines may be continued to as many terminal lines as necessary.

- The characters listed in Table 2.6 have special meanings in CFS requests.

* In the examples in this manual, a \ is used.

Table 2.6. Characters with special meanings in CFS requests

| Character | Meaning |
|---|---|
| , | A comma separates CFS requests on the same line. |
| \ | A backslash at the end of a line of characters indicates that the next line is a continuation of the request or series of requests. CFS does not replace the \ with a space. |
| # | A pound sign at the end of a line of characters functions in the same way as a \. |
| " " | Quotation marks delimit input text for the **info=** keyword. To continue the text from line to line, use the \. A <RETURN> without the \ ends a quotation. |
| ( ) | Parentheses enclose multiple values that are assigned to a single keyword. |
| - | When used as the value of a keyword, a minus sign indicates that the system default is to be used rather than a user-defined value.<br>When used as a path parameter, a minus sign specifies that the path stored in the default working directory is to be used.<br>When used as a keyset parameter, a minus sign specifies that your userid (Cray User Number) is to be used as the keyset name.<br>When used as the value of the password field in a validation entry, a minus sign signifies that no password is required. |
| < > | Angle brackets delimit a comment contained in a request line. A comment may not be embedded in a parameter or a keyword and its associated values. |
| : | A colon associates a workerfile with a CFS path. For example, **save a:b** specifies that the UNICOS file **a** is to be saved on CFS with the name **b**. The name preceding the colon must always be the workerfile name; that following the colon, the CFS file name. |
| ' | The prime notation (*n'*) specifies that the working directory **dir***n* is to be used, where *n* is a number from 0 to 9. The notation also signals the use of the associated password **pw***n* where relevant. |
| / | A slash is the first character of a complete path and precedes each node name that is part of a path. |

## 2.3 Responses

CFS attempts to execute each request you enter, and returns a response for each request (unless the request follows an incorrect request in a multiple-request line). In most cases, the response is from the CFS program; in some cases, the origin of the response is **cfsi**.

When a request has been successfully executed, a *normal* response is returned. When CFS detects an error and cannot execute a request, an *error* response is returned, indicating some reason for the failure.

- The basic format for a normal response is

  ```
  000 date time command workerfile:path
  ```

  where *command* and *workerfile:path* are the command, workerfile (if any), and path (if any) indicated in the corresponding request. The command and path may be spelled out in full, even if you entered them in abbreviated form. If your request involves a file transfer, the name of the workerfile and the number of bits transferred (in decimal) are included in the response.

  The commands **list**, **show**, **status**, and **set** (alone or with standalone keywords) are used to request information. The normal response for each of these provides the desired information in an appropriate format.

- The basic format for an error response is

  *****warning:** (Y XXXX $Z_1$, $Z_2$, ...)yy/dd/mm hh:mm text

  where

| | |
|---|---|
| **warning** | may sometimes be replaced by the word **fatal**, such as when the CFS program is stopped temporarily for some reason or communication with **cfsi** has been lost and must be restarted. |
| Y | is a severity code for use within CFS. |
| XXXX | is an error code that may be used by controllers on the Cray. |

| | |
|---|---|
| `z`$_1$`, z`$_2$`, ...` | are status bits that may be present and that may tell more about the type of error. |
| `yy/mm/dd` | is the date the message was issued, where *yy* is the year, *mm* the month, and *dd* the day. |
| `hh:mm` | is the time the message was issued, where *hh* is the hour and *mm* the minute. |
| `text` | is a short message telling what happened. If the message is continued over more than one line, continuation lines begin with a single asterisk (*) in column 2. |

The error and severity codes and any status bits can generally be ignored because all the information needed to spot and correct a problem will be contained in the text message. These messages are usually self-explanatory and easy to understand; Programming Assistance can help with any that are unclear.

## 2.4 Single-Line Execution

In single-line execution, you include the **cfsi** call and all the requests in a single execution line. When you enter a <RETURN> at the end of the line, all of the requests on the line are executed and **cfsi** returns you to the UNICOS system prompt.

For single-line execution, use the following format

**cfsi** *request1, request2, ...*

where each *request* is the equivalent of the standard request line introduced in Section 2.2. The space following **cfsi** and the commas between requests are required.

Use a backslash (\) or a pound sign (#) to continue an execute line that is too long to enter as one line on the terminal screen (see Section 2.2.5).

## 2.5 Using cfsi in the Background and in Batch Jobs

Using `cfsi` in the background and in batch jobs is basically the same as using `cfsi` interactively. However, you may find it helpful to note the following:

- Single-line execution works in the background and in batch mode. To terminate a single-line process you have submitted in the background, use the UNICOS commands `ps` and `kill`.
- You can use the continuation characters \ and # in batch jobs.
- The comment field in a request can be helpful in documenting scripts.
- The `cfsi` wait and retry cycle that functions when CFS or a CFS network connection is down allows a batch job to remain in a wait state for up to about an hour. If you do not want a batch job or a job submitted to run in the background to wait that long, you may want to include in your shell script an escape based on system-produced status messages. Alternatively, you may want to include a `status` request early in the job and base the decision about whether to run on the response to that request.

# 3. File Organization on CFS

CFS uses a hierarchical file storage structure called a tree.* When you create a tree, you provide the framework that allows CFS to locate your files when access is requested and to maintain pertinent information regarding the files.

Simple and complex tree structures are the subject of Section 3.1. Section 3.2 explains *paths*, which are the roadmaps CFS needs to find specific locations in a tree. Finally, Section 3.3 provides a brief introduction to the basic commands needed to work with complex tree structures.

## 3.1 Tree Structures

A tree structure on CFS consists of named segments called *nodes* connected by *branches* that indicate the hierarchy. As described in more detail later in this section, nodes are special areas where information relevant to the access of your files is stored. Examples of tree structures are shown in Figures 3.1 and 3.2 on pages 3-4 and 3-6. By convention, CFS trees are drawn in inverted form.

There are two types of nodes: *directory nodes* and *file descriptor nodes*. In turn, directory nodes are of two types: *root nodes* and *subdirectory nodes*.

The origin of a tree structure is called the *root node, root directory*, or *root*. It is represented by a triangle in tree-structure diagrams and is the first *level* of organization. The tree in Figure 3.2 originates in a root node named 010203.

Branching out from the root node are *descendant nodes*, which may be either

- subdirectory nodes (*subdirectories*), which are represented in diagrams by rectangles and can also have descendants, or
- file descriptor nodes, which are represented by circles and cannot have descendants.

* The structure is similar to that provided under VMS and UNICOS.

The direct descendants of the root node represent the second level of the tree. In Figure 3.2, there are two second-level descendants, **heatcode** and **reports**. In this case, both are subdirectory nodes. The direct descendants of second-level subdirectories form the third level of the tree, etc. In Figure 3.2, **data** is a subdirectory at the third level of organization and the descendant files **set1** and **set2** are at the fourth level. You can use up to 48 levels of organization; 6 or 7 is a practical limit.* Note that because CFS trees "grow down," the lower the node is located in a tree, the higher its level number will be.

The *parent* of a node is connected to it by a branch and is at the next higher level in the tree. A root node has no parent. A *subtree* is any node in a tree and all of its descendants at all levels. Thus, in Figure 3.2, **reports** is the parent of **monthly** and **quarterly**. The subtree originating in **reports** includes **monthly**, **jan**, **feb**, and **quarterly**.

You use requests to create directory nodes that establish the storage groupings and relationships you want for your files. CFS creates a file descriptor node, in the tree position you indicate in your request, for each new file you create on the system. Each file descriptor node will have the same name as its corresponding file and will contain a pointer to the actual current storage location of the file.

As already noted, the nodes of a tree are storage areas for information relevant to access of the files associated with the tree. The information is categorized (i.e., stored in specific *fields*); the fields for each type of node differ somewhat.

- Some of the stored information is for use by CFS only. An example is the file storage location pointer associated with a file descriptor node.

* Note that the total number of characters in the path to a node (see Section 3.2) cannot exceed 96.

- Some of the node information is contained in named fields that you can display using the `list` command with various list options (see Chapter 9, Section 9.10 and Chapter 11, Section 11.9). For example, the *descendants* field for a given directory node contains a list of all the node's immediate descendants. CFS uses the list to help locate descendant files; you can display the list if you want to view that portion of your tree structure (using the `list` command with the `lo=d` option).

Some node fields contain information (e.g., the time a file was last accessed, the file size, the list of a directory's direct descendants) that is updated automatically by CFS as necessary. Other fields contain default values (usually null values) or values specified by the user in requests using certain keywords and commands. For example, the *info* field for a subdirectory node will contain the default, a null, unless you insert descriptive information (e.g., using the `info=` keyword with the `add` command as described in Chapter 11, Section 11.7).

The contents of the node fields are known as *node attributes*. For more information about node fields/attributes, see Chapter 11, Section 11.9 and Appendix A.

### 3.1.1 More about root nodes

You must create a root node before you can store any files on CFS. There are two types of root nodes:

- *userid root*: a root with your personal user identifier for a name (your Cray User Number). This root name is reserved for you and may be the only one you need to use.

  Create your userid root node by entering the following request

  ```
  create
  ```

  Once you have created your userid root, it becomes the default (i.e., CFS automatically goes to that directory when you invoke the `cfsi` program).

- *named root*: a root with a name you define. The name must be unique to the *entire* CFS system. CFS will return an error response if you attempt to create a node with a name that is not unique.

  Create a named root node by entering

  ```
  create root
  ```

  where *root* is the name you want to assign to the new root. The name must start with an alphabetic character, be no longer than 16 characters in length, and incorporate only the characters

  `a-z, A-Z, 0-9, $, %, *, +, -, .,` and `_`.

See Chapter 9, Section 9.4 for a procedure for listing the names of your root nodes.

### 3.1.2 Simple tree structures

If you want to store only a few files on CFS, you can probably use your userid root and a simple tree structure (one with no subdirectory nodes) as shown in Figure 3.1. In this figure, the root node is the parent of five file descriptor nodes; the file descriptor nodes are the direct descendants of the root node. These relationships are indicated by branches connecting each file node to the root node.

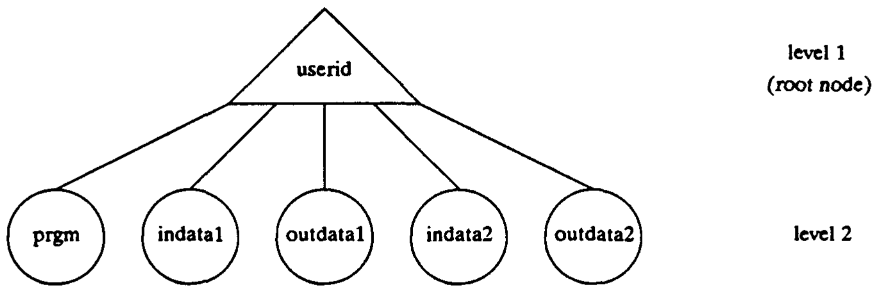


**Fig. 3.1. A simple tree structure.**

You may store files under your userid root node by entering

```
save filename
```

where *filename* is the name of a file in your current working directory on UNICOS and also the name the file will have on CFS. To retrieve a copy of the file, enter

```
get filename
```

To delete a file from CFS, enter

```
delete filename
```

To see a list of the files in your userid root directory, enter

```
list lo=d
```

### 3.1.3 Complex tree structures

For numerous files, you may find it helpful to use a complex tree, grouping related files under subdirectories as shown in Figure 3.2. In this example, the user whose userid root node name is `010203` has grouped the files related to the program `heatcode` in one subdirectory and his monthly and quarterly reports in another subdirectory. Creating separate subdirectories in the `heatcode` subtree for each version of `heatcode` and for the data to be input to both versions is a further aid to easy location of the associated files.

You should provide meaningful names for the nodes you create. Doing this will greatly assist you in determining what each file contains without having to transfer it to a worker system for examination. Figure 3.2 is an example of the use of meaningful node names. Compare this figure to Figure 3.3, which pictures the same tree structure with node names that leave you guessing as to file content.

To work with complex trees, you must understand paths, which are described in Section 3.2. An introduction to the basic commands you will need to work with complex tree structures is provided in Section 3.3.

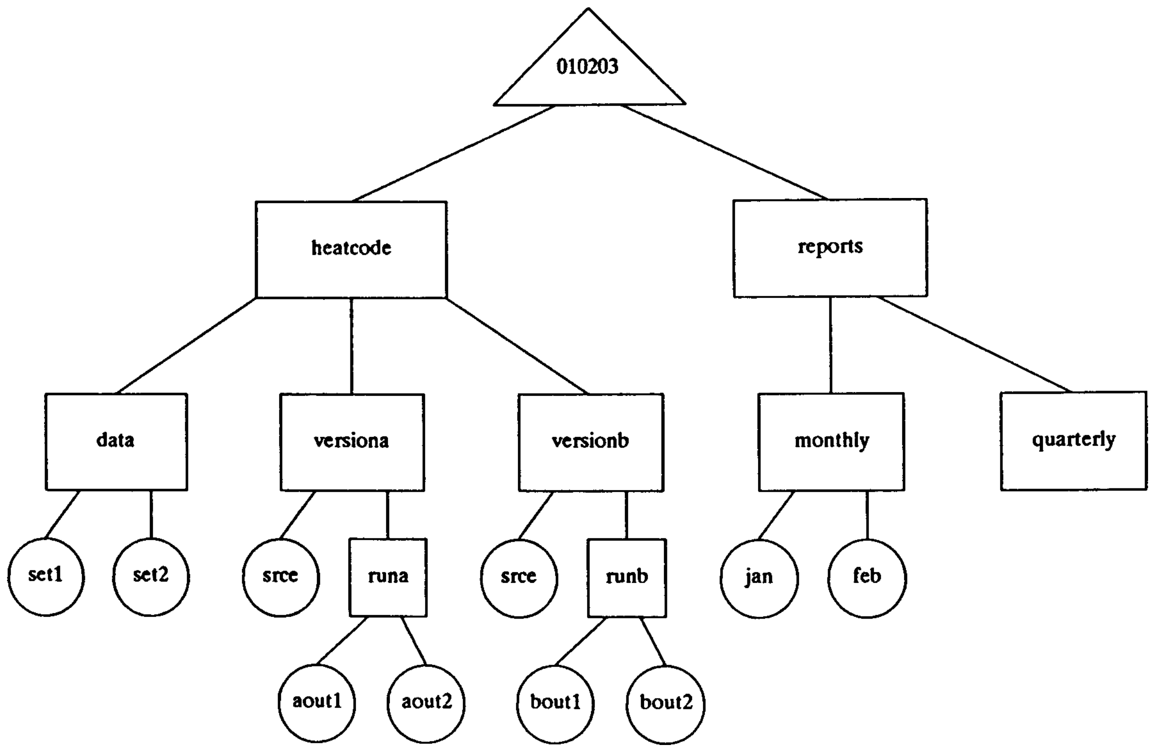


**Fig. 3.2. A complex tree structure with node names that provide guidance as to file content.**

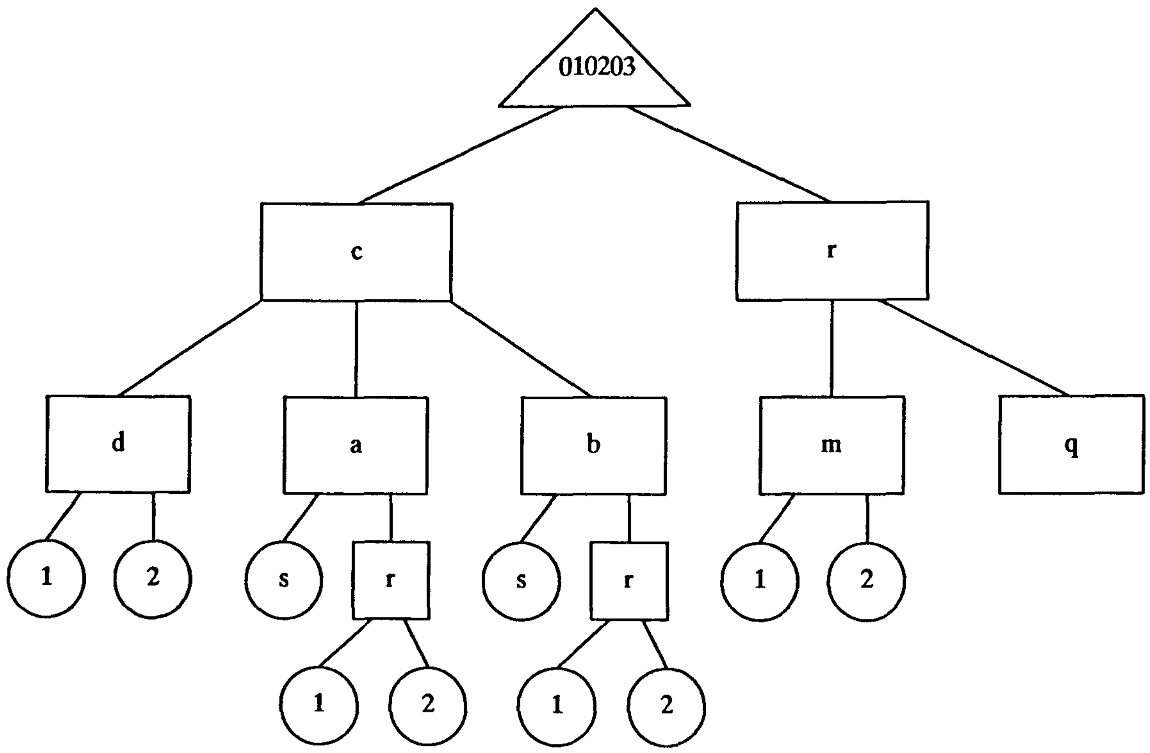


Fig. 3.3. A complex tree structure with poorly named nodes.

### 3.1.4 Useful information about root nodes and tree structures

If your projects require more than one charge number, you must use multiple storage trees (i.e., create named roots in addition to your userid root), because each tree can have only one charge number associated with it. Chapter 5 contains information about CFS charges.

You may want to create named root nodes if you have sets of files you want to share with other users. For an example, see Chapter 8, Section 8.2.

There is no charge for root and subdirectory nodes. You can have as many trees in CFS as needed.

Should your projects change in size or scope, you can reorganize tree structures. The **move** and **modify** commands, described in Chapter 9, are useful for this purpose.

A root node must have a name unique on CFS. Two or more non-root nodes may have the same name as long as they are not immediate descendants of the same parent node. It is the *complete path* for the node, not the node name itself, that must be unique (see Section 3.2).

You cannot view the contents of a file stored on CFS without first copying the file to UNICOS. Therefore, it is helpful to use meaningful file names and to place identifying information in the *info* fields of nodes (see Chapter 11, Sections 11.7 and 11.13).

## 3.2 Paths: Specifying a Node as a Request Parameter

To refer to a node as a parameter in a CFS request (see Chapter 2, Section 2.2.2), you must give CFS the *complete path* (also referred to as the *complete path name*) for the node. This means that you must provide the sequence of node names that describes to CFS the route to follow through the tree to reach the specific node. For example, the complete path for the file **aout1**, which contains the first set of output data obtained from running Version A of **heatcode** in Figure 3.2, is

```
/010203/heatcode/versiona/runa/aout1
```

The complete path for the userid root node 010203 is

```
/010203
```

The following rules apply when specifying complete paths:

- A complete path must be preceded by a slash (/).
- Each node name must be separated from the next by a slash.
- The root node must be the first node entered, followed by the appropriate second-level node, etc.
- The maximum length for a path, including slashes, is 96 characters.
- You may specify a path in another user's area if that user has granted you the necessary access rights to the node defined by the path (see Chapter 8).
- There are three options for specifying complete paths in a request.
  - — You can enter the path in full with a leading slash as shown in the **heatcode** example above. Any path beginning with a slash is taken by CFS to be a complete path.
  - — You can enter a partial path with no leading slash and have CFS retrieve the leading nodes of the path from a *working directory* (see Chapter 6).
  - — You can tell CFS to retrieve a complete path for a node from a working directory (see Chapter 6).

  CFS provides ten working directories for your use. They are not directories in the same sense as roots or subdirectory nodes, but simply areas to store complete or partial paths. Your default working directory **dir0**, by default, contains the path for your userid root node (i.e., */userid*). Using working directories can simplify the process of entering path names associated with complex tree structures.

If you are using only a userid root node and a simple tree structure to store unclassified files, a knowledge of working directories is not necessary (i.e., you can skip Chapter 6). You need only be aware that when you enter a file name as the parameter for a request, you are entering a partial path (no leading slash). CFS automatically appends the contents of `dir0` and a slash to form the complete path. Thus, CFS reads the request

**`save`** *filename*

as

**`save`** */userid/filename*

You can use either form of the parameter in a request, but taking advantage of the default working directory means fewer characters to type.

## 3.3 Building a Complex Tree Structure

To build a complex tree structure, you will need to use the following commands, which are described in more detail in Chapter 9:

| | |
|---|---|
| **`create`** | The first step in building a complex tree structure is to create a root directory as described in Section 3.1.1. |
| **`add`** | Use the **`add`** command to add subdirectory nodes to the root node and to other subdirectory nodes. Unless you use working directories, you must enter as the request parameter the complete path for each subdirectory to be added; the name of the subdirectory will be the last node name in the path. |
| **`save`**, **`store`** | Use the **`save`** or **`store`** command to add file descriptor nodes as files are transferred to CFS. Unless you use working directories, you must enter as the request parameter the complete CFS path for each file; the name of the file will be the last node name in the path. |

For example, you can use the following set of requests to build the tree structure shown in Figure 3.4:

```
create reports88
add /reports88/monthly
add /reports88/quarterly
save /reports88/monthly/jan
save /reports88/monthly/feb
save /reports88/monthly/mar
save /reports88/quarterly/first
```

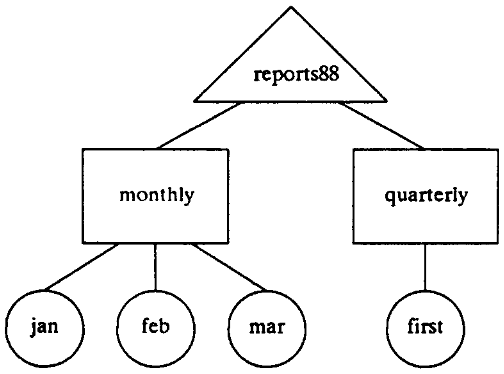


**Fig. 3.4. A complex tree structure.**

The commands **copy**, **delete**, **move**, **remove**, and **modify** can affect tree structures. These commands are discussed in Chapter 9.

# 4. A Tutorial to Get You Started

This chapter contains an eleven-step tutorial to introduce you to use of **cfsi** on the UNICOS Cray. The following conventions apply:

- **User input**: Literal user input is indicated in boldface monospaced type, user-specific input in italic type.
- **System responses**: System responses are in regular monospaced type, except for user-specific information, which is in italics.

Representative responses are shown, but will vary in specifics from those you receive. UNICOS and **cfsi** prompts are not shown.

1. Go to the UNICOS directory in which you want to work and create a file called **test**. UNICOS is your worker system, and **test** is a workerfile.
2. Invoke **cfsi**

```
  cfsi
connected to cfs concentrator, Z number is userid
000 89/11/06 17:48 cfs interface started (+00108)
```

3. Your userid on the Cray is your six-character Cray user number. Create your userid root node by entering

```
  create  /userid
000 89/11/06 17:48 create /userid
```

   (The slash is optional.) Alternatively, because CFS accepts a blank parameter with **create**, you can simply enter **create**; CFS will obtain your userid from the default working directory **dir0**, which, by default, contains */userid.*

4. Confirm that the root has been created by listing information about the node using list option **a**. You can specify the parameter as shown or use a blank parameter, which, as with **create** above, will cause CFS to insert the contents of **dir0**.

```
list /userid lo=a

    node name:  userid
    node type: root directory
oid:
charge:  000nnnnn
space-time product starts: 89/10/30 10:54
online space:            0.00 megabytes
online files:       0
online space-time product:              0.0 megabyte days
offline space:           0.00 megabytes
offline files:      0
offline space-time product:             0.0 megabyte days
```

Because you did not specify a charge number using the **charge=** keyword when you created the root node, the *charge* field for the node contains the default charge number for your UNICOS session. All charges associated with files stored in the root directory tree will be made to that number.

5. Save a copy of the workerfile `test` on CFS under your userid root.

```
  save test
000 89/11/06 17:48 save test:/userid/test
001 (5784 bits)
```

Because the parameter you supply (i.e., **test**) is an incomplete path, CFS automatically appends the contents of **dir0** and a slash (i.e., */userid/*) to form the required complete path */userid/***test**.

6. Display a list of the direct descendants of your userid root using a blank parameter.

```
list lo=d
   node name:  userid
   node type: root directory
descendants:
test
```

You could have omitted **lo=d** because it is the default for directory nodes.

7. Attempt to delete your userid root. Note that CFS will not accept a blank parameter with `remove`.

```
    remove /userid
***warning:  (1 0104) 89/11/06 17:49 : attempted to remove
 * directory node which has one or more descendants.
```

Because the directory is not empty (it contains the descendant `test`), CFS does not execute the request and returns an error response.

8. Retrieve a copy of the file `test` from CFS; the copy will replace the identical version currently on UNICOS in the directory from which you invoked `cfsi`.

```
    get test
000 89/11/06 17:49 get test:/userid/test
001 (5784 bits)  last written 89/11/06 17:48
```

9. Delete the copy of `test` that resides on CFS.

```
    delete test
000 89/11/06 17:49 delete /userid/test
```

10. Delete your now-empty userid root.

```
    remove /userid
000 89/11/06 17:49 remove /userid
```

You are now back to where you were in Step 2.

11. Terminate your `cfsi` session.

```
end
```

# 5. Charges for Use of CFS

You must provide at least one valid five-digit C&TD computer charge number (also known as a C&TD request number) to which your use of CFS can be billed. Billing is done on a monthly basis.

This chapter describes how charges are calculated, how you specify the charge number(s) you want to use, and how you can determine what storage charges you have accumulated to date during a monthly accounting period.

## 5.1 How CFS Charges are Determined

Each of your root nodes has an associated charge number. To view the current charge number for the root node defined by *path*, enter

```
list path lo=a
```

and refer to the *charge* field in the response.

The number listed in the *charge* field is billed for the storage of all descendant files, as follows:

- a file charge (also known as a directory charge): a small fee for each file you have in storage at the end of each monthly accounting period.
- a storage space charge: a charge based on the size of the files stored and the length of time they were stored (a space-time product). The unit of measure is the megabyte (Mb) month. Your CFS files may be stored on tape (offline) or disk (online). To check the storage type device for a file, enter the following request, where *path* is the complete path for the file, and look at the *storage type* field of the response.

  ```
  list path lo=g
  ```

For the current CFS rates, see `man charges` on UNICOS.

## 5.2 Specifying Charge Numbers

If you are working on projects that have different charge numbers, you must create a root node for each project and store your files accordingly.

Use the **charge=** keyword with the **create** command to assign a charge number to a new node. For example, the following request creates a root node named **project** containing the charge number 12345 in the *charge* field. CFS charges for files in the associated tree will be assessed to this number.

```
create project charge=00012345
```

If you omit the **charge=** keyword when you create a root node, as when you create your userid root by entering

```
create
```

the default account number for your UNICOS session will be assigned to the node.

You can change the charge number associated with one of your root nodes by entering

**modify** *path* **ncharge=000***xxxxx*

where *path* is the path for the root and *xxxxx* is the new charge number. The new number will go into effect immediately.

## 5.3 Calculating Interim CFS Charges

File storage costs are the major component of a monthly bill. You can figure your file storage charges to date for a particular root node. An example demonstrating the procedure is shown below.

- User ABC wants to determine the charges he has accumulated for the current accounting period for files stored under his root named **project**. As a first step he enters the following request to list accounting information for **project**. Accounting information is updated at least every 24 hours.

  ```
  list /project lo=a
  ```

CFS returns the following response:

```
node name: project
node type: root directory
oid:
charge: 00012345
space time product starts: 89/10/30 10:54
online space: 134.959 megabytes
online files: 43
online space time product: 1572.8 megabyte days
offline space: 4.615 megabytes
offline files: 80
offline space time product: 928.6 megabyte days
```

The *space time product starts* field indicates the beginning of the current accounting period.

- ABC refers to the `man` file `charges` and finds that the rates listed below apply. Note: these rates may not correspond to actual current rates.
  - file charge: $ .055 per file
  - online (disk) storage: $1.00 per Mb month
  - offline (tape) storage: $1.00 per Mb month

  The charge per Mb month is actually based on a 30-day month. The rate for shorter accounting periods will be proportionately less and for longer periods, proportionately more.

- ABC then figures his storage charges to date for the month for files stored under the root `project`.
  - $1.00/Mb month ÷ 30 Mb days/Mb month = $0.0333/Mb day
  - cost for online storage:

    1572.8 Mb days × $0.0333/Mb day= $52.37
  - cost for offline storage:

    928.6 Mb days × $0.0333/Mb day= $30.92
  - total storage cost to date: $52.37 + $30.92 = $83.29

ABC, in the above example, can obtain a tentative file charge for the month as follows:

43 online files + 80 offline files = 123 files
123 files x $ .055 per file = $6.77

However, the number of files he will have in storage at the end of the month is likely to differ from any interim total.

# 6. Working Directories

To identify a file or directory node to CFS in a request (i.e., to specify a root or path as a parameter), you must supply a complete path as described in Chapter 3. Doing this is easy if you are working with a simple tree structure such as the one shown in Figure 3.1 on page 3-4. To retrieve a copy of the file **indata1**, shown in that figure, you have to enter only

```
get /userid/indata1
```

However, when you have organized your files into a complex structure such as the one shown in Figure 6.1 on page 6-7, entering paths can be quite tedious. For example, to retrieve a copy of **aout1**, shown in that figure, the following is required:

```
get /userid/heatcode/versiona/runa/aout1
```

CFS provides the mechanism of *working directories* to simplify the task of entering paths with commands that require a path parameter (see Table 2.3).

*Working directories are not to be confused with tree structure directories. A working directory is simply an area in which to store a path.* After you store a path in a working directory, you can refer to the directory in requests where use of working directories is permitted; CFS will automatically substitute the path contained in the directory.

The first three sections of this chapter describe the most commonly used features of working directories: Section 6.1 deals with the names and contents of working directories; Section 6.2 describes how to use working directories; and a series of examples is provided in Section 6.3.

If the node defined by a given path is protected by a password (see Chapter 8), you must supply that password in any request that accesses the node. For each working directory there is a correspondingly named area in which the password for the stored path can be placed for automatic retrieval by CFS. Passwords stored in this way are known as *working directory passwords* and are described in Section 6.4.

Note that a working directory password does not supply a password for access to the corresponding working directory as the name seems to indicate, but rather to the path contained in the working directory.

## 6.1 The Names and Contents of Working Directories

There are ten working directories, **dir***n*, where *n* is one of the digits 0 through 9. **dir0**, the default, can be abbreviated as **dir**, **d0**, or **d**; the others as **d***n*. Each working directory is represented in your active keyset (see Chapter 3, Section 3.2 and Chapter 7) by the corresponding **dir***n*= keyword.

Each working directory can contain one path at any given time. Working directories containing paths that originate with a root node are most frequently used and so are the only ones considered in this manual.

Each time you invoke **cfsi**, **dir0** contains the complete path for your userid root (i.e., */userid*) and **dir1** through **dir9** contain null values.

You can replace the contents of any working directory, including **dir0**, with a new path using the corresponding **dir***n*= keyword in a request. To store a path in a working directory for

- the remainder of your current CFS session or until changed again, use the **set** command with a keyword and value **dir***n*=*path* as in

  **set dir2=***/userid***/reports**

  After execution of this request, the current value for **dir2=** in your active keyset is */userid*/**reports**.

- the duration of a single request, use **dir***n*=*path* with a command other than **set** as in the following example

  **delete first second dir0=***/userid***/reports**

  As explained in Section 6.2.1, this use of **dir0=** causes */userid*/**reports/** to be appended automatically to the partial paths **first** and **second**. After execution of the request, the value for **dir0=** in your active keyset returns to what it was prior to execution.

To restore the contents of a working directory to the default value, use the **set** command with the **dir***n***=** keyword set to a minus sign (-). For example

```
set dir0=-
```

replaces the present contents of **dir0** with */userid* and

```
set dir1=-
```

replaces the current contents of **dir1** with a null value.

To view the contents of a working directory (i.e., the value of a **dir***n***=** keyword in your active keyset), enter

```
set dirn
```

where *n* is the appropriate working directory number.

> If you begin to receive unexpected responses (particulary **node does not exist**) when entering CFS requests, check the values set for the working directories you are using to be sure those values are what you currently need.

Paths you store as part of your active keyset during a **cfsi** session are not saved when you exit the session. If you intend to use one or more paths over a period of time, you may want to store them as a part of a saved keyset for long-term use (see Chapter 7).

## 6.2 Using Working Directories

Use of the default working directory **dir0** is discussed in Section 6.2.1; use of **dir1** through **dir9** is the subject of Section 6.2.2.

### 6.2.1 Using DIR0

Four rules govern use of your default working directory `dir0`.

1. If you omit the path parameter (i.e., use a null parameter) in a request with `list` or the root parameter in a request with `create`, CFS will automatically insert the path currently saved in `dir0` (i.e., use the current active keyset value for `dir0=`). Two examples follow:

   — Assuming that `dir0` contains the default path */userid*, you can list accounting information about your userid root node by entering

   ```
   list lo=a
   ```

   — Assuming that `dir0` contains the path */userid*/`reports`, you can list the descendants of the subdirectory node `reports` by entering

   ```
   list
   ```

   Other commands that require a path parameter will not accept a null. With those commands, you must enter a path in full or make use of a working directory in one of the ways indicated in Rules 2 through 4.

2. For commands that require a path parameter, CFS automatically appends the contents of the default `dir0` and a slash to any partial path in a request unless you provide other instructions. The following examples illustrate:

   — If you invoke `cfsi` then enter

   ```
   save a b c d
   ```

   CFS makes use of the default value for `dir0=` in your active keyset and executes

   ```
   save /userid/a /userid/b /userid/c \
   /userid/d
   ```

   — If you then enter

   ```
   save a b c d \
   dir0=/userid/reports/monthly
   ```

CFS makes use of the value for **dir0=** specified in the request and executes

```
save /userid/reports/monthly/a \
/userid/reports/monthly/b \
/userid/reports/monthly/c \
/userid/reports/monthly/d
```

to save files **a**, **b**, **c**, and **d** under the subdirectory defined by the path */userid*/**reports/monthly**. The value of **dir0=** in your active keyset is restored to */userid* after execution of the request.

— The following two requests are an alternative to the above procedure:

```
set dir0=/userid/reports/monthly
save a b c d
```

The value of **dir0=** in your active keyset, in this case, does not revert to */userid* after execution of the **save** request.

3. A minus sign (-) used as the parameter in a request refers specifically to the default working directory **dir0**. Two examples follow:

   — To retrieve a copy of the file **book1**, which is a direct descendant of the root named **library**, you can enter the two requests shown below.

   ```
   set dir0=/library/book1
   get -
   ```

   — Using the value just set for **dir0**, create a copy of the file **book1** under your userid root and call the copy **another**.

   ```
   copy - to /userid/another
   ```

4. The shorthand notation *n'* described in Section 6.2.2 also applies to **dir0**.

### 6.2.2 Using DIR1 through DIR9

Use the *n prime notation* (*n'*), where *n* is the number of the directory, to tell CFS what working directory to copy a path from.

The *n'* notation used alone tells CFS to use the path stored in `dirn` as the complete path. For example, if you enter

```
set dir4=/userid/summary
```

then enter

```
get 4'
```

CFS executes

```
get /userid/summary
```

Or, if you want to store a value in `dir4` only for the duration of the request, you can enter

```
get 4' dir4=/userid/summary
```

To tell CFS to append the contents of `dirn` to a partial path, enter *n'path* as in the following example:

```
get 5'jan
```

If `dir5` contains the path */userid/*`reports`, a copy of the file defined by the path */userid/*`reports/jan` is retrieved. You also can retrieve a copy of the file `jan` by entering

```
get 5'jan dir5=/userid/reports
```

## 6.3 Examples

The following serial examples demonstrate how to place paths in working directories and how to use those paths in CFS requests. The examples are based on the tree structure shown in Figure 6.1 on the next page.

- Place the path */userid/*`heatcode/versiona` in `dir3` for the remainder of the current CFS session or until another request stores a different path in `dir3`.

  ```
  set dir3=/userid/heatcode/versiona
  ```

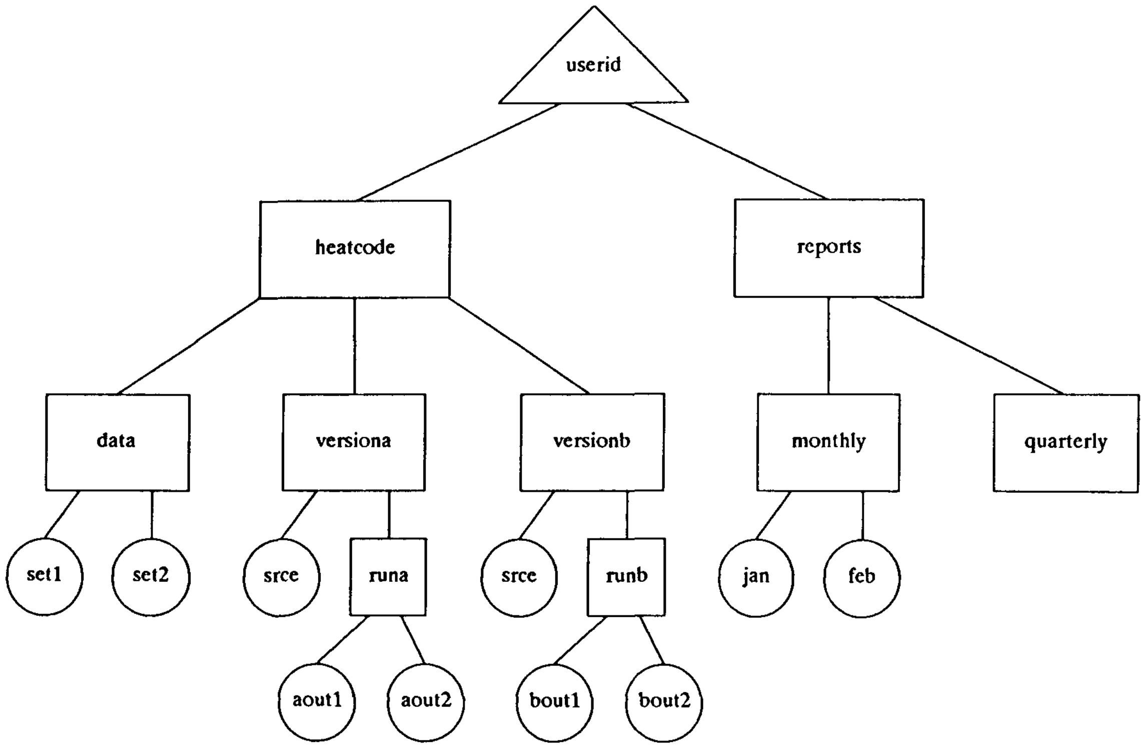


**Fig. 6.1. A complex tree structure.**

- Now, move a copy of `aout1` to UNICOS using the `3'` notation to tell CFS to append the contents of `dir3` to the partial path that follows.

  ```
  get 3'runa/aout1
  ```

  If you had chosen not to make use of `dir3`, the following request would have been required:

  ```
  get /userid/heatcode/versiona/runa/aout1
  ```

- Use the `3'` notation and a partial path to place the path */userid*/`heatcode/versiona/runa` in `dir6` for the remainder of the current CFS session or until another request stores a different path in `dir6`.

  ```
  set dir6=3'runa
  ```

- Move a copy of `aout2` to UNICOS using `dir6`.

  ```
  get 6'aout2
  ```

- Place the path */userid*/`reports/monthly` in your default working directory

  ```
  set dir0=/userid/reports/monthly
  ```

  so that you can retrieve copies of `jan` and `feb` by entering

  ```
  get jan feb
  ```

  instead of

  ```
  get /userid/reports/monthly/jan \
  /userid/reports/monthly/feb
  ```

  If additional monthly reports are present and have to be retrieved, this use of `dir0` will be even more advantageous.

- Add a new subdirectory called **feedback** under the subdirectory **monthly**, making use of the present contents of **dir0**.

  ```
  add feedback
  ```

  The following request could also have been used:

  ```
  add 0'feedback
  ```

- Place a copy of the file **aout1** under the subdirectory **monthly**, making use of the present contents of **dir0** and **dir6**.

  ```
  copy 6'aout1 to aout1
  ```

  Another abbreviated request could also have been used.

  ```
  copy 6'aout1 to 0'aout1
  ```

- Restore the contents of **dir0** to the default value (i.e., */userid*).

  ```
  set dir0=-
  ```

- Make it easy to use the file **jan** in several requests by placing the complete path for the file in **dir0**

  ```
  set dir0=/userid/reports/monthly/jan
  ```

  where it can be used as the default path in commands such as

  ```
  get -
  ```

  which copies **jan** to UNICOS,

  ```
  copy - to /userid/jan
  ```

  which places a copy of **jan** under your root node, and

  ```
  delete -
  ```

  which deletes the copy of **jan** that is located in the **monthly** subdirectory.

- Set the value of `dir1=` to */userid/*`heatcode/data` for the duration of one request to simplify deletion of the files `set1` and `set2`.

```
delete dir1=/userid/heatcode/data 1'set1 \
1'set2
```

## 6.4 Working Directory Passwords

Working directory passwords are discussed in this chapter because their use is linked to the use of working directories. If you do not intend to share CFS files with other users, you do not need to be concerned with this section. If you do intend to share files, you should read Chapter 8 before proceeding with this section.

If a directory or file node is protected by a password, any user to whom the password applies must supply that password in each request that accesses the node.

When using passwords under `cfsi`, you must enter a caret (^) before the password. Be sure to protect your screen from observation by others when passwords are displayed.

For every working directory (`dir0` through `dir9`), a corresponding space (`pw0` through `pw9`) is provided to store the password required to access *the path stored in that directory*. A password stored in one of these locations is called a working directory password.*

* Despite what the term *working directory password* seems to imply, the password has nothing to do with access to the working directory, itself.

To view a stored password, enter

```
set pwn lpw=on, set lpw=off
```

where *n* is the appropriate digit.

The password stored in **pw0** is the default. Just as **dir0** can be shortened to **dir**, **pw0** can be expressed as **pw**.

Each time you invoke **cfsi**, each of the keywords **pw0=** through **pw9=** is set to a null value in your active keyset. As with a **dir***n*= keyword, you can store a new **pw***n*= value either for the duration of a request or for the remainder of your session.

- To store a working directory password for the duration of a request, include a **pw***n*= keyword in a request with a command other than **set**. The password stored temporarily replaces the current value for the **pw***n*= keyword in your active keyset; the current value is restored after execution of the request.

  The most common use of a temporarily stored password is in requests in which the path parameter is entered in full. In such cases, use **pw=** to supply the password as shown in the following example:

  ```
  get /userid/experiments/testrun1 pw=^views
  ```

  Similarly, use **pw=** when part of the path is supplied using **dir0**.

  The following examples typify other common uses of temporarily stored passwords:

  — Assuming that the password **views** is required for access to the directory **experiments**, a direct descendant of your userid root, you can enter

  ```
  get dir6=/userid/experiments \
  pw6=^views 6'testrun2 6'testrun3 \
  6'testrun4
  ```

  to retrieve copies of the files **testrun2**, **testrun3**, and **testrun4** from the subdirectory **experiments**. Note that the working directory **dir6** (and thus **pw6**) was chosen arbitrarily; any other working directory could have been used.

— Assuming that the password **enter** is required for access to the subdirectory **ideas**, a direct descendant of your userid root, you can enter the following request to save copies of the UNICOS files **new**, **old**, and **super** under **ideas**:

```
save dir6=/userid/ideas pw6=^enter \
6'new 6'old 6'super
```

As in the previous example, **dir6** was chosen arbitrarily.

- To store a working directory password for the remainder of your session or until changed again, use the **set** command with a keyword and value *pwn=password* as shown below.

```
set pw6=^views
```

The current value for **pw6=** in your active keyset is now **views**.

Generally, you will want to store a path and the associated password at the same time as in the example that follows:

```
set pw6=^views dir6=/userid/experiments
```

Once you have stored a password using **set**, CFS automatically uses that password whenever you refer to the corresponding working directory in a request. Thus, after entering the above request, you can enter requests that access **experiments** without any reference to **views**. For example:

```
get 6'testrun1
get 6'testrun2
get 6'testrun3
save test2:6'testrun2
list lo=g 6'
```

You should note three special situations concerning the use of working directory passwords.

1. If a request contains a complete path that requires a password for access, and you do not supply a password in the request as in

   ```
   get /userid/experiments/testrun1
   ```

   CFS tries the password, if any, currently set for **pw0=** in your active keyset.

2. If you use a working directory to supply part of a path in a request, you must use the corresponding **pw*n*=** keyword to supply the password associated with the path even if the password is not required for access to the portion of the path stored in the working directory.

   For example, assume that **dir4** contains the path */userid/***oldfiles**, for which no password is required for access; **oldfiles** has a subdirectory, **final**, for which the password **access** has been specified; and you want to save **file1** under **final**. Do this with

   ```
   set pw4=^access
   save 4'final/file1
   ```

   or

   ```
   save 4'final/file1 pw4=^access
   ```

3. The **move** and **copy** commands require a parameter consisting of two paths (i.e., *path1* **to** *path2*). Either path or both paths may require a password for access. If these passwords are different (or if only one path has a password), at least one of the paths must use the *n'* format to distinguish which password is associated with which path.

   For example, assume that your default working directory **dir0** contains the path */userid/***programs** and that you have set **pw0=** to **enter**, the password required to access that path. You want to move the file **test** from the subdirectory **programs** to the subdirectory **newsave**, which has the complete path **/revisions/newsave** and the password **evalz**. Two ways to accomplish this follow. (Note that if **revisions/newsave** belongs to another user, that user must have granted you either write or insert access rights to **newsave** for the **move** request to be executed; access rights are described in Chapter 8, Section 8.2.3.)

```
move test to 3'test dir3=/revisions/newsave \
pw3=^evalz
```

or

```
set pw3=^evalz dir3=/revisions/newsave
move test to 3'test
```

The following request would not work

```
move test to /revisions/newsave/test \
pw=^evalz
```

because CFS would take the use of **pw=^evalz** to be a temporary setting for **pw0**. Thus, it would think you were supplying the wrong password for */userid*/**programs** and omitting the necessary password for **/revisions/newsave**.

# 7. Keysets

A set of all keywords with their assigned values is called a *keyset.**

Keywords are introduced in Chapter 2, Section 2.2.3. Individual keywords and their value options are described in Chapter 11.

There are two types of keysets. *Active keysets* are discussed in Section 7.1; *saved keysets* are introduced in Sections 7.2 and 7.3. All users make use of active keysets; use of saved keysets is optional. Knowing how to work with keysets can help make entering CFS requests faster.

Other users cannot use your keysets.

* Values for the keywords **`maval=`**, **`mcval=`**, **`mdval=`**, **`mnval=`**, and **`muval=`**, which are used only to change the values of other keywords and are not covered in this manual, are not included.

## 7.1 Active Keysets

When you invoke `cfsi`, an *active keyset* is created with each keyword set to its default value (see Table 7.1 on page 7-3). You can change the value of a keyword in your active keyset using the **set** command as follows:

**set keyword=***value*

Specifying a keyword and value in a request with another command simply replaces the value of the keyword in your active keyset for the duration of the request. In this manual, the **set keyword=***value* request is used only with `dir`*n*`=`, **pw***n***=**, and **lpw=**. For other keywords, where a value other than the default is required, that value is defined in the request (i.e., is set temporarily).

Each command other than **set** has a limited set of keywords associated with it. A complete list for each command, including keywords not described in this manual, is shown in Table 7.2 on pages 7-4 and 7-5. You can assign a value to any of those keywords in a request with the command; CFS automatically gets the values for any of those keywords not defined in the request from your active keyset. Thus, when you enter

**list** */userid* **lo=u lpw=on**

CFS uses the values for **lo=** and **lpw=** shown and takes the values for the other associated keywords, **dir0=**, **pw0=**, and **talk=**, from your active keyset. Assuming that those values are the defaults, the validation entries associated with your userid root node will be listed with passwords displayed.

This manual does not cover the use of all keywords and, thus, describes the use of only some of the possible options for each command (see Table 7.3 on page 7-6). Unless you require unusual functionality, the default values for the other keywords should meet your needs and those keywords should not be of concern.

**Table 7.1. Default values for the keywords described in this manual**

| Keyword | Default Value |
|---|---|
| `aval=` | null |
| `charge=` | default number |
| `cval=` | null |
| `dir0=` | userid root |
| `dir1=`, ..., `dir9=` | null |
| `dval=` | null |
| `grp=` | null (CFS chooses) |
| `info=` | null |
| `kpw=` | null |
| `lo=` | `g` for a file<br>`d` for a directory |
| `lpw=` | `off` |
| `ncharge=` | null |
| `ngrp=` | null |
| `ninfo=` | null |
| `nname=` | null |
| `nrel=` | null |
| `nuse=` | null |
| `pwn=` | null |
| `setgrp=` | `off` |
| `so=` | `n` or `names` |

Table 7.2. Commands and all their associated keywords

| Command | Keyword[a] | | | | |
|---|---|---|---|---|---|
| add | cl= | dir*n*= | info= | muval= | pw*n*= |
| | talk= | uval= | | | |
| adopt | kpw= | talk= | | | |
| copy | charge= | dir*n*= | mnval= | ngrp= | ninfo= |
| | nrel= | nuse= | nval= | pw*n*= | talk= |
| create | charge= | cl= | dir*n*= | info= | muval= |
| | oid= | pw*n*= | talk= | uval= | |
| delete | dir*n*= | pw*n*= | talk= | | |
| end | none | | | | |
| free | kpw= | talk= | | | |
| get | charge= | dir*n*= | fm= | pw*n*= | talk= |
| keep | kpw= | info= | talk= | | |
| list | dir*n*= | lo= | lpw= | pw*n*= | talk= |
| modify | aval= | cval= | dir*n*= | dval= | maval= |
| | mcval= | mdval= | ncharge= | ncl= | ninfo= |
| | nmaster= | nname= | noid= | npart= | nrel= |
| | nuse= | pw*n*= | setgrp= | talk= | |
| move | dir*n*= | ninfo= | pw*n*= | talk= | |
| remove | dir*n*= | pw*n*= | talk= | | |
| replace | charge= | comp= | dir*n*= | fm= | info= |
| | nrel= | pw*n*= | talk= | | |
| save | charge= | comp= | dir*n*= | fm= | grp= |
| | info= | muval= | pw*n*= | rel= | talk= |
| | use= | uval= | | | |

Table 7.2. Commands and all their associated keywords (continued)

| Command | Keyword[a] | | | | |
|---|---|---|---|---|---|
| set | aval= | charge= | cl= | comp= | cval= |
| | dir*n*= | dval= | fm= | grp= | info= |
| | kpw= | lo= | lpw= | maval= | mcval= |
| | mdval= | mnval= | muval= | ncharge= | ncl= |
| | ngrp= | ninfo= | nmaster= | nname= | noid= |
| | npart= | nrel= | nuse= | nval= | oid= |
| | pw*n*= | rel= | setgrp= | so= | talk= |
| | use= | uval= | | | |
| show | kpw= | lpw= | so= | talk= | |
| status | talk= | | | | |
| store | charge= | comp= | dir*n*= | fm= | grp= |
| | info= | muval= | nrel= | pw*n*= | rel= |
| | talk= | use= | uval= | | |

[a] Some of these keywords are not described in this manual, and many are used only in a limited way. If you require capabilities that go beyond the scope of this manual, refer to the *Common File System CFS Interface Reference* for more information. Copies of that manual are available for reference in the Programming Assistance offices.

**Table 7.3. Commands with the associated keywords covered in this manual**

| Command | Keywords | | | |
|---|---|---|---|---|
| add | dir*n*= | info= | pw*n*= | |
| adopt | kpw= | | | |
| copy | dir*n*= | ngrp= | pw*n*= | |
| create | charge= | info= | pw*n*= | |
| delete | dir*n*= | pw*n*= | | |
| end | | | | |
| free | kpw= | | | |
| get | dir*n*= | pw*n*= | | |
| keep | kpw= | | | |
| list | dir*n*= | lo= | lpw= | pw*n*= |
| modify | aval= | cval= | dir*n*= | dval= |
| | ncharge= | ninfo= | nname= | |
| | nrel= | nuse= | pw*n*= | setgrp= |
| move | dir*n*= | pw*n*= | | |
| remove | dir*n*= | pw*n*= | | |
| replace | dir*n*= | pw*n*= | | |
| save | dir*n*= | grp= | pw*n*= | |
| set[a] | dir*n*= | lpw= | pw*n*= | |
| show | kpw= | lpw= | so= | |
| status | | | | |
| store | dir*n*= | grp= | pw*n*= | |

[a] Any keyword may be used in standalone form with **set**.

Using the **set** command to assign **dir***n*= and **pw***n*= values can save you time if you will be referring frequently to one or more nodes during a **cfsi** session. For more information, see Chapter 6.

To display the keywords and their values for your active keyset, enter

```
set
```

or

```
set -
```

To display the value of a particular keyword in your active keyset, enter

**set** *keyword*

where *keyword* is the keyword without an = sign and value (i.e., a standalone keyword).

## 7.2 Saved Keysets

A saved keyset is a copy of an active keyset you have given a name and stored for future use. The name can be from 1 to 16 characters in length and incorporate any of the following characters:

**a-z, A-Z, 0-9, $, %, *, +, –, ., ** and **_**.

When you save a copy of your current active keyset, the active keyset remains unchanged.

You may have as many saved keysets as needed. However, unless you make significant changes in your active keyset and intend to use the altered values frequently, saving a copy of a keyset for later use may not be worthwhile.

Your saved keysets cannot be used by other users.

You have the option of protecting a keyset you save with a keyset password. However, because you are the only one who can access any of your saved keysets, use of keyset passwords on an unclassified system is of marginal value. For more information about keyset passwords, see Chapter 8, Section 8.3.

To specify a keyset password when you save a keyset and to supply the password to access the keyset later, use the `kpw=` keyword. When using a keyset password in a request, you must enter a caret (^) before the password.

You can create a saved keyset using the **`keep`** command as shown below, first for a keyset that will not have an associated password and then for one that will. The parameter *keyset* is a keyset name of your choice.

**`keep`** *keyset* or **`keep`** *keyset* kpw=^*password*

The saved keyset will be a copy of your current active keyset. If a keyset with the name you specify already exists, it is overwritten.

Once a keyset has been saved, it remains available from session to session and can be used repeatedly until written over by another **`keep`** request or deleted with the **`free`** command as shown below, where *keyset* is the name of the saved keyset:

**`free`** *keyset* or **`free`** *keyset* kpw=^*password*

Use the **`adopt`** command, as follows, to retrieve a copy of a saved keyset and make it the current active keyset, where *keyset* is the name of the saved keyset:

**`adopt`** *keyset* or **`adopt`** *keyset* kpw=^*password*

You may store a saved keyset that has your userid as its name by entering

**`keep -`** or **`keep -`** kpw=^*password*

To retrieve that keyset, enter

**`adopt -`** or **`adopt -`** kpw=^*password*

To display the values of a saved keyset use the **`show`** command as described in Chapter 9, Section 9.17.

## 7.3 An Example of the Use of Saved Keysets

The example that makes up the remainder of this section demonstrates, in a limited way, the use of saved keysets.

Assume that you will be working in two subdirectories during this and later sessions. One subdirectory is defined by the path */userid/*`reports/monthly`, the other by */userid/*`project`.

To make your work as easy as possible, you decide to use saved keysets. After invoking **cfsi**, you modify the initial active keyset using the **set** command as follows:

```
set dir0=/userid/project
```

You then save a copy of this keyset, giving it your userid as its name.

```
keep -
```

Your active keyset still has **dir0=** set to */userid/*`project`. You next make the modification you want to suit your needs for working in the subdirectory **monthly**.

```
set dir0=/userid/reports/monthly
```

and save a copy of this keyset, giving it the name **reptkey**.

```
keep reptkey
```

You want to save a file under **monthly**, so you use the current value for **dir0=** in your active keyset and enter

```
save filename
```

You next decide to store a file under `project`. To do this, all you have to enter is

```
adopt -
save filename
```

Finally, you decide to store another file under **monthly**, which is done easily as follows:

```
adopt reptkey
save filename
```

Because you will be storing files under `monthly` and `project` during future sessions, you retain your saved keysets (i.e., do not free them) so that they can be adopted when needed.

# 8. Security and File Sharing

Security on CFS has three components:*

- classification level,
- access rights, and
- password protection.

The UNICOS Cray is an unclassified, non-sensitive system.** Thus, all directories and files on the K-25 CFS have a *classification level* of unclassified (i.e., the *cl* field for each CFS node contains a u). You must not store files of another classification level on UNICOS or CFS.

*Access rights* are used to define what type of access a user will have to a particular node. By default, you are the only user with access rights to your CFS nodes. If you want to share your files with other users, you must grant them access rights to the appropriate nodes.

*Passwords* are used to help ensure that access is limited to the users to whom you have granted access rights. Use of passwords is optional on an unclassified system.

Access rights that are granted and passwords that are required are specified using *master user* (owner) and *user validation entries*. An entry written for a specific node is stored, as appropriate, in the *master user validation* or *user validations* field of the node. Once a password is associated with a user in a validation entry that applies to a node, the user must supply the password in each request that accesses the node.

Access rights and passwords are covered as components of validation entries in Section 8.1. Section 8.2 provides examples of their use in file sharing.

* See Appendix D for information about the file protection assumed by a file copied from CFS to UNICOS.

** For security information, see Chapter 6 of *A Guide to Computing at Energy Systems*, which can be ordered through DO MANUAL on the C&TD PDP-10 or by calling 4-0331.

You can use keyset passwords with saved keysets. For instructions, refer to Section 8.3.

For information about the commands mentioned in this chapter, see Chapter 9; individual keywords are described in Chapter 11.

## 8.1 Validation Entries

A general description of validation entries is presented in Section 8.1.1. Further information about the components of validation entries is provided in Sections 8.1.2 through 8.1.4, as follows:

| | |
|---|---|
| Section 8.1.2 | Access rights accumulation modifiers |
| Section 8.1.3 | Access rights |
| Section 8.1.4 | Passwords |

For examples of how to use validation entries to allow file sharing and provide protection for your files, see Section 8.2.

### 8.1.1 General description

Validation entries determine who will have access to your files and what the limit of the access will be.

A validation entry consists of four fields separated by slashes (/) as shown below

*userid/rights/password/modifier*

where

*userid* is the userid (Cray User Number) of a specific user for whom rights are being granted.

*access rights* are one or more single-letter abbreviations representing the extent of the privileges the specified user will have for the node; the choices are `r`(ead), `w`(rite), `m`(odify), `i`(nsert), `b`(estow), `e`(xecute), and `a`(ppend) as defined in Section 8.1.3. Possession of one access right does not imply that you have others. For example, you may have write access without having read access. A null (no character or space between the slashes delimiting the field) is valid; if the access rights field is a null, as in the example below,

```
010203//-/s
```

any access privileges granted to the user at a higher level in the tree are removed for this node and any of its descendants (how access rights set for one node in a tree affect a lower level of the tree is explained later in this section).

*password* is a password of five to eight alphanumeric characters required of the specified user to exercise the access rights granted in the entry (see Section 8.1.4). A minus sign (-) in the field signifies that no password is required. For any node but a root, a null password field (no character or space between the slashes delimiting the field) indicates that the password from a higher level in the tree is to be used. When entering a password as part of a request, you must enter a caret (^) before the password.

*modifier* is the *access rights accumulation modifier*. The three options (**set**, **and**, and **or**) are described in Section 8.1.2.

If you have modify access, you can view the validation entries for a node that does not require a password by entering

```
list path lo=u
```

and checking the *validation* field(s) in the response. If any of the validation entries contain passwords, the passwords will be displayed as strings of percent signs (i.e., **%%%%%%%%**); to view those passwords, you must include **lpw=on** in the request.

```
list path lo=u lpw=on
```

For a node that requires a password for your access, the following request causes validation entries, including passwords, to be displayed for the node defined by *path*.

```
list path lo=u lpw=on pw=^password
```

Some basic rules that apply to validation entries follow:

1. Access rights are assigned and passwords specified by writing validation entries for nodes.

   — You, by default, have full access to your root nodes and all their descendants. Each of your nodes has a master user validation entry that describes your access rights to the node.

   — User validation entries must be written to allow other users access to your CFS nodes. CFS allows only one entry per userid in the *user validations* field of a node.

2. In general, the access rights you assign to a user by writing a validation entry for one node automatically apply also to all nodes of its subtree (the descendant nodes are said to inherit the validation of the parent). If you want to change the access for the user to a descendant node in the tree, you must write an appropriate validation entry for that node; the access granted by the entry will be passed on to the descendants of that node.

   The same rules apply to passwords and access rights modifiers.

   See Chapter 9, Sections 9.3 and 9.12 for the rules that determine the validation entries that apply to new files created using the `copy` command and files that have been repositioned using `move`.

3. When you attempt to access a directory or file descriptor node, CFS examines all the validation entries for you on the specified path. Whether you are allowed to access the node depends on

   — what access rights are accumulated along the path and

   — how the rights are affected by the specified access rights accumulation modifiers.

4. You can use `list` with `lo=u` to view the validation entries for a node (modify access to the node required). The response is as follows:

— User validation entries

A user validation entry is displayed only in the *user validations* field of the node for which it was written. It does not appear in the *user validations* fields of descendant nodes that inherit the validation.

— Master user validation entries

A master user validation entry is displayed in the *master user validation* field for every node, whether written for that node or inherited from a parent node. For example,

```
node name: project
node type: file descriptor
master user validation:
userid/rewaibm/-/s
user validations:
```

If a change has been made to the master validation entry for any non-root node, duplicate entries for the owner's userid are displayed, one in the *master user validation* field and one in the *user validations* field. For example,

```
node name: project
node type: file descriptor
master user validation:
userid/reaibm/-/s
user validations:
userid/reaibm/-/s
```

5. As implied by 2 and 4 above, you may have to check the protection of the parent or a higher node to determine the access a specific user has to a given node.

6. You should set validations at as high a level as possible in a tree structure and usually for a directory node rather than a file descriptor node.

Modify access to a node allows a user to write (i.e., create and change) and delete master user and user validation entries for the node. Bestow access allows a user to write entries that grant one or more of those rights possessed by that user. For example, if user 010203 has read and bestow access to a node, he can write the entry `040506/r/-/s` but not the entry `040506/rw/-/s`.

The procedures for creating, changing, and deleting validation entries are described below. For simplicity, passwords are not specified and the access rights modifier `s` is used in all cases. The access rights field is used to demonstrate entry changes; however, the same procedures apply to the password and access rights modifier fields.

- **Creating a validation entry**
  - Master user validation entries

    When you create a root node, CFS creates a master user validation entry in the *master user validation* field for that node (see Chapter 3 and Appendix A for information about fields). For example, if you enter

    ```
    create
    ```

    to create your userid root, your master validation for the root will be

    *userid*`/rewaibm/-/s`

    The entry gives you, as owner, full access rights to the root. The entry is passed on to all descendant nodes, unless you make changes as described under the next bullet in this list. You can include a password in the entry when you create the root as described in Section 8.1.4.
  - User validation entries

    Add these entries after the node has been created. To do so, use the `aval=` keyword with the `modify` command. For example, enter

    `modify` */userid* `aval=010203/r/-/s`

    to grant user 010203 read access to your userid root.

    The *user validations* field for your userid root now contains the entry `010203/r/-/s`.

- **Changing validation entries**

  — Master user validation entries

  Use the `cval=` keyword with the `modify` command to change the master user validation entry for a root node. For example, to remove your write access to your userid root, enter

  ```
  modify /userid cval=userid/reaibm/-/s
  ```

  CFS will proceed through the directory tree automatically making the same change to the master user entries for all descendant nodes. If a node is encountered for which a different entry has already been created, this automatic process will not be applied to that node or any of its descendants. CFS will not allow the owner's modify access to a node to be removed.

  The procedure for changing the master user validation entry for a descendant node depends on whether a change has previously been made for that node. If the change is the initial change, you must use the `aval=` keyword with the `modify` command. For example, assume the entry for your userid root is */userid*/`rewaibm/-/s` and that you save the file `project` under the root. The root's master user entry is automatically passed on to `project`, as you can see by entering

  ```
  list project lo=u
  ```

  to view the following response:

  ```
  node name: project
  node type: file descriptor
  master user validation:
  userid/rewaibm/-/s
  user validations:
  ```

Now, you decide to protect the file from accidental deletion by removing your write access to the file descriptor node. To do this, use **aval=** as shown below.

```
modify project aval=*userid*/reaibm/-/s
```

The list command above will now produce

```
node name: project
node type: file descriptor
master user validation:
userid/reaibm/-/s
user validations:
userid/reaibm/-/s
```

Suppose you decide to delete `project`. Before doing this, you must reinstate your write access. Because this is not the initial change to the validation entry (i.e., there is an entry for your userid in both the *master user* and *user validation* fields for the node), you must make the change using `cval=` with `modify` as shown below.

```
modify project cval=*userid*/rewaibm/-/s
```

Any change made to your master user validation entry for a subdirectory node is passed on to subsequently created descendant nodes.

— User validation entries

To change a user validation entry for a node, use the **cval=** keyword with the `modify` command. For example, if user 010203 has read access to the second-level subdirectory node **calendar** and you want to grant him write access in addition, enter

```
modify calendar cval=010203/rw/-/s
```

to replace the validation entry `010203/r/-/s` with `010203/rw/-/s`.

- **Deleting a validation entry**

  Use **dval=** with **modify** to delete a validation entry for a node. Set the value of **dval=** to the userid of the user whose entry is to be deleted. For example, enter

  ```
  modify calendar dval=010203
  ```

  to delete the entry for user 10203 from the node **calendar**.

  CFS will not permit you to delete a master user validation entry from a root node. If you attempt to delete the master user validation entry from a non-root node, the current entry will be replaced with the entry in effect for the parent node.

## 8.1.2 Access rights accumulation modifiers

At each node in the path, any access rights assigned to you in a validation entry are combined with rights assigned at earlier nodes in the tree according to the access rights accumulation modifier (i.e., *modifier*) specified in the validation entry for the node. (Note that combining rights may result in rights being added or removed.) The three possible modifiers **set**, **or**, and **and**, represented respectively by the letters **s**, **o**, and **a**, are described below. To avoid the confusion that can result from many access settings on many nodes, use **s** whenever possible.

| | |
|---|---|
| **set (s)** | The user has only those access rights specified at the current node. Access rights set at a higher level in the tree are no longer valid for the specified node or its descendants. |
| **or (o)** | The user has all access rights in effect down to this node plus any assigned in this node. **or** is used to add additional rights. |
| **and (a)** | The user has only those access rights that were previously assigned and are also assigned at the current node. **and** is used to further limit access. Use of **and** can result in eliminating all access rights. |

### 8.1.3 Access rights

Any combination of the following seven access rights can be included in a validation entry. Use the single-character abbreviation shown in parentheses to specify the associated right.

- **Read (r) access**

  For a file descriptor node, read access allows a user to retrieve the associated file from CFS using the **get** command, to specify the file as the source in a **copy** request (modify access also required if the *user validations* field for the node contains one or more entries), and to display node information using **list** with the **lo=** keyword set to **d**, **g**, **i**, or **s**.

  For a root or subdirectory node, read access allows a user to display node information using the **list** command with **lo=** set to **a**, **d**, or **g**.

- **Write (w) access**

  Caution: Write is a very powerful access right. In general, you should not grant other users write access to your root nodes. Grant write access to other nodes only when specific requirements cannot be satisfied by insert access.

  For a file descriptor node, write access allows a user to use the **replace**, **delete**, and **store** (when a **replace** results) commands. It also permits using **modify** with the keywords **nname=**, **nrel=**, and **ninfo=** to change the file name, release date, and contents of the *info* field for the node.

  For a root or subdirectory node, write access allows a user to delete the node using the **remove** command and to use **modify** with **ninfo=** and **nname=**. It also permits adding descendant subdirectory and file nodes using **add** and **save** (or **store** when a new file descriptor node is being created).

  For CFS to execute a user's **copy** or **move** request, the user must have write or insert access to the destination parent directory node.

- **Modify (m) access**

  Caution: Modify is a very powerful access right. In general, you should not grant other users modify access to your root nodes. Grant modify access to other nodes only when specific requirements cannot be satisfied by bestow access.

  Modify access, when granted for a directory or file descriptor node, allows a user to list validation entries for the node using the `list` command with `lo=u` and to add, change, and delete validation entries using `modify` with the `aval=`, `cval=`, and `dval=` keywords, respectively.

  The owner of a root must always have modify access to every node of the associated tree; CFS will not execute a request that attempts to remove the owner's modify access.

  In addition to permitting a user to alter validation entries, modify access allows a user to change other node attributes using the `modify` command with the appropriate keyword as shown in the table below.

  | Keyword | Type of Node |
  |---|---|
  | `ncharge=` | Root |
  | `ninfo=` | All |
  | `nname=` | All |
  | `nrel=` | File descriptor |
  | `nuse=` | File descriptor |
  | `setgrp=` | File descriptor |

  Modify access allows a user to use the node or subtree as the source in a `move` request.

  Modify access to the source file in a `copy` request is required in addition to read access when the *user validations* field for the node contains one or more entries.

- **Insert (i) access**

  Insert access is a limited form of write access that applies only to directory nodes. It allows subdirectory and file descriptor nodes to be added (using `add` and `save` or `store`, respectively) but not replaced, removed, or deleted from the directory to which the user has this access.

  Having insert access to a directory node also permits a user to specify that node as the destination parent in a `copy` or `move` request.

- **Bestow (b) access**

  Bestow access allows a user to add user validation entries to a directory or file descriptor node using `modify` with `aval=`. The user may assign only those access rights he or she possesses.

- **Execute (e) access**

  The intended function of execute access has not yet been implemented. This access right currently allows a user to retrieve a copy of a CFS file using the `get` command, duplicating a function allowed by read access. Generally, do not remove this access type from an owner validation entry and do not include it in validation entries for other users.

- **Append (a) access**

  Not currently available. Generally, do not remove this access type from an owner validation entry and do not include it in validation entries for other users.

A few functions are allowed by more than one type of access right. A summary of these is presented in Table 8.1.

**Table 8.1. Functions permitted by more than one access right**

| Function | Node Type | Access Rights |
|---|---|---|
| Use `modify` with `ninfo=`, `nrel=`, and `nname=` | file desc. | write<br>modify |
| Use `modify` with `ninfo=` and `nname=` | directory | write<br>modify |
| Specify the node as the destination parent in a **move** or copy request | directory | write<br>insert |
| Add subdirectory and file descriptor nodes using **add**, **save**, and **store** | directory | write<br>insert |
| Add user validation entries using **aval=** with `modify` | directory<br>file desc. | modify<br>bestow[a] |
| Retrieve a copy of a CFS file using **get** | file desc. | read<br>execute[b] |

[a] With bestow, only those access rights possessed may be granted.
[b] Use read.

## 8.1.4 Passwords

The *password* field of the validation entry for a user may contain

- a minus sign (-), indicating that no password is required;
- a null (i.e., no characters or space between the delimiting slashes), indicating that the password assigned to the user at a higher level on the tree is required; or
- a string of five to eight alphanumeric characters.

If you specify a password when you write a validation entry for another user, you must tell the user what that password is. If you specify a password for your own access, be sure to take precautions not to forget it.

If you forget a password required for you to access your own CFS nodes, call the Programmer Aide at your site:

| | |
|---|---|
| K-25 | 4-1060 |
| X-10 | 4-5400 |
| Y-12 | 4-8376 |

If a validation entry for a user for a node *specifies* a password (i.e., contains a null or a string of characters), the user must *supply* that password in any request that accesses the node. The password is also required for the user's access to any descendant nodes unless a change is made lower in the tree. For information about how to use working directory passwords to supply passwords in requests, see Chapter 6, Section 6.4 and Chapter 11, Section 11.19. Note: if you supply a password when none is required, you will receive an error response.

To specify a password for yourself when you create a root, use the **pw=** keyword as shown in the example below.

```
create reports pw=^sesame
```

Otherwise, to add, change, and delete password specifications, follow the procedures for manipulating validation entries described in Section 8.1.1. For examples of these procedures involving passwords, see Section 8.2.

Note that a node may have more than one associated password. For example, the password **sesame** may be required for you to access your root named **reports** and the password **newpass** for user 010203 to have read access to the same node. You can use the following requests to make this arrangement:

```
create reports pw=^sesame
modify /reports pw=^sesame \
aval=010203/r/^newpass/s
```

When passwords are displayed, you must protect your screen from observation by others.

## 8.2 File Sharing Examples

Examples of four different file sharing arrangements are provided in this section. The procedures followed in each case may not be the only ones possible.

The first three examples are based on the tree structure illustrated in Figure 8.1.

The access rights accumulation modifier **s** is used in all cases (as recommended in Section 8.1.2). This modifier indicates that any rights granted for the specified user at a higher level on the tree are replaced by the rights specified at the current node.

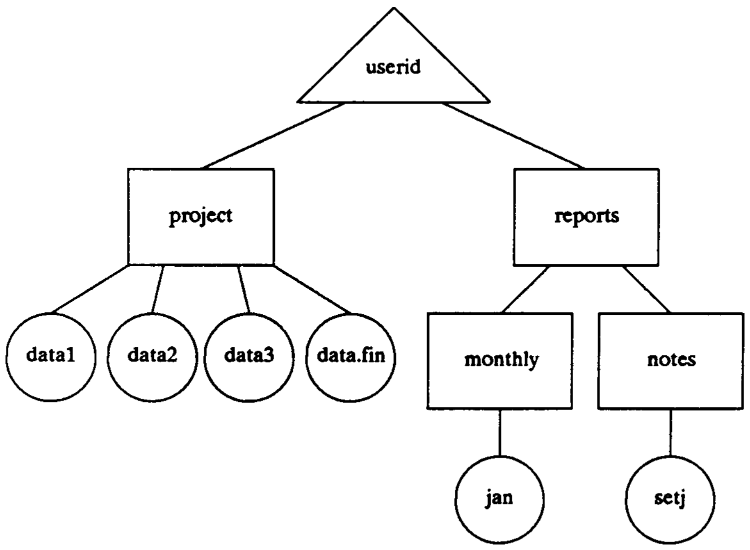


**Fig. 8.1. Tree structure for Examples 1 through 3 in Section 8.2.**

- **Example 1 (refer to Figure 8.1):**

  You have specified no passwords for access to your userid root or to any of its descendants. You want your supervisor (user 010203) to have read access to any files stored under **monthly**, but not to the personal evaluations stored under **notes**.

  *Option 1:*

  Enter

  ```
  modify /userid/reports/monthly \
  aval=010203/r/-/s
  ```

  to add the specified validation entry to the node **monthly**. The minus sign in the password field of the entry indicates that user 010203 will not have to supply a password for access.

  When user 010203 enters

  ```
  get /userid/reports/monthly/jan
  ```

  CFS examines the validation entries for user 010203 along the path for **jan**. Although no access is granted (i.e., there are no applicable validation entries) to the root or to **reports**, the entry for **monthly** permits the necessary access to that node and its descendants. The request will be executed. Note that the validation entry could have been added, instead, to the file descriptor node **jan**. However, you would have had to add the same validation entry for each file (i.e., **feb**, **mar**, . . .) to which you wanted to permit the access.

  If user 010203 enters

  ```
  delete /userid/reports/monthly/jan
  ```

  the request will be denied because write access is required to delete a file.

  If user 010203 enters

  ```
  get /userid/reports/notes/setj
  ```

  CFS will find no applicable validation entries along the specified path (i.e., no access has been granted) and will not execute the request.

*Option 2:*

If you want, you can place the access permission in the node **reports** by entering

```
modify /userid/reports \
aval=010203/r/-/s
```

However, you would then have to block the access of user 010203 to **notes** and its subtree. You can do this by using a null access rights field as shown below.

```
modify /userid/reports/notes \
aval=010203//-/s
```

Then, when user 010203 enters

```
get /userid/reports/notes/setj
```

CFS will find that, although the node **reports** contains a validation entry that grants read access, the next node in the path, **notes**, contains an entry that removes that right. The request will not be executed.

- **Example 2 (refer to Figure 8.1):**

  This example is the same as Example 1 except that you specified the password **allow** for yourself when you created your userid root by entering

  ```
  create pw=^allow
  ```

  Enter

  ```
  modify /userid/reports/monthly pw=^allow \
  aval=010203/r/^allow/s
  ```

  to add the validation entry granting read access for user 010203 to the node **monthly**. In this case, you also require user 010203 to supply the password **allow** when accessing **monthly** and its descendants.

  User 010203 can retrieve a copy of **jan** by entering

  ```
  get /userid/reports/monthly/jan pw=^allow
  ```

- **Example 3 (refer to Figure 8.1):**

  You have specified the password **permit** for your own access to your userid root. You want to grant read access to the three members (users 010203, 040506, and 070809) of a project team to all the **data** files under **project** except for **data.fin**; they will have both read and insert access to that file. You do not want to allow access to anyone but yourself to **reports** and its descendants. You decide not to share your password but rather to require that the project members use the password **newpass**.

  To grant read access for the three users to **project** and its descendants, enter

  ```
  modify /userid/project pw=^permit \
  aval=(010203/r/^newpass/s 040506/r/^newpass/s \
  070809/r/^newpass/s)
  ```

  Then, to add insert access for **data.fin**, enter

  ```
  modify /userid/project/data.fin pw=^permit \
  aval=(010203/ri//s 040506/ri//s 070809/ri//s)
  ```

  The null password fields indicate that the password that is required for access to **project** (i.e., **newpass**) is also required to access **data.fin**. The modifier **s** indicates that the rights granted in the entries for the node **data.fin** supercede any previous rights.

  User 070809 leaves the project and must no longer have access to the **data** files. You remove that access by entering the following two requests:

  ```
  modify /userid/project/data.fin pw=^permit \
  dval=070809

  modify /userid/project pw=^permit \
  dval=070809
  ```

- **Example 4:**

  You create the root **data** to accumulate the results of experiments being done by several members of a project team you are supervising. These members have the userids 010203, 040506, and 070809. You want these users to be able to add files to, but not delete files from, the directory. You require that everyone accessing the node supply the password **results**. To create the root and place the chosen password in the *master user validation* field for the node, enter

```
create data pw=^results
```

Then enter

```
modify /data pw=^results \
aval=(010203/i/^results/s 040506/i/^results/s \
070809/i/^results/s)
```

to assign to your colleagues insert access and specify the password **results**.

User 040506 can then save his data file called **exp1** under **data** by entering

```
save /data/exp1 pw=^results
```

or copy a data file defined by the path **/040506/exp2** (which requires the password **techn** for access) to **data** by entering

```
set dir1=/040506/exp2 pw1=^techn
copy 1' to 2' dir2=/data/exp2 \
pw2=^results
```

If user 040506 tries to delete **exp2** as follows,

```
delete /data/exp2 pw=^results
```

the request will be denied on the basis of insufficient access rights.

## 8.3 Keyset Passwords

When you create a saved keyset using the **keep** command (see Chapter 7, Section 7.2), you can protect that keyset with a keyset password using the **kpw=** keyword. For example, the following request saves your active keyset as the saved keyset **newkey** and specifies the password **keypass**

```
keep newkey kpw=^keypass
```

which must then be supplied using the kpw= keyword whenever you refer to the keyset in an **adopt**, or **free** request such as

```
free newkey kpw=^keypass
```

However, the value of establishing a keyset password is marginal because you are the only user who can access your saved keysets.

If you do specify a password for a saved keyset, the password can be any combination of from five to eight alphanumeric characters. When you use a keyset password in a request, you must enter a caret (^) before the password.

# 9. Command Descriptions

A command indicates the action to be taken by CFS.

Each of the commands introduced in Chapter 2 is described in more detail in this chapter. The command descriptions are arranged in alphabetical order and follow the format outlined below. For each description, outline topics that do not apply are omitted. The descriptions cover only selected features; for additional features, see the *Common File System CFS Interface Reference*, which is available in the Programming Assistance offices.

- **Command name.**
- **Functional description.**
- **Shortest command abbreviation:** The shortest acceptable command abbreviation, if abbreviation is allowed.
- **Parameter(s) required:** Parameter options, when a parameter is required. A series of three elipses (...) indicates that multiples of the associated parameter are allowed. For more information about parameters, including alternative ways of specifying paths, see Chapter 10.
- **Keywords and their values:** Keywords, if any, suggested for use with the command. If you do not specify a value for an applicable keyword in a request with the command, CFS automatically uses the value set in your active keyset. For general information about keywords and keysets, see Chapters 2 and 7. For more details about individual keywords and their values, see Chapter 11.

  For most commands, most of the keywords listed relate to exercising options or defining conditions for the request. To define or redefine a node attribute (i.e., set the contents of one of a node's information fields), you will usually have to use the `modify` command with one or more of its keyword options.

- **Security considerations:** Notes concerning access rights required to execute the command and, in special cases, information about how CFS handles validation entries when executing the command. For general information about validation entries, passwords, and access rights, see Chapter 8.
- **Additional comments:** Additional comments, if any.
- **Examples:** The first example always demonstrates the most common, no-frills use of the command. In all cases, two conditions are assumed unless otherwise stated:

  — **`dir0`**, the default working directory, contains your userid root (i.e., **`/userid`**) and

  — the nodes involved are not password-protected.

  In addition, if a request copies a file to or from a UNICOS directory, it is assumed that you invoked **`cfsi`** from that directory. See Appendix C for information about how to specify other UNICOS directories in CFS requests.

  The request shown in an example may not be the only one that can be used to accomplish the described function.

  As explained in Chapter 2, Section 2.2.5, a backslash (\) at the end of an input line indicates that the request is continued on the next input line.

If you just want to store a few files in your userid root, the following commands are the ones you are most likely to need:

```
create   delete  end   get
list     status  save
```

The first example shown in each of the corresponding command descriptions is generally the only one you will need to understand.

You may want to use the following tables in Chapter 2 for quick reference:

| Table | Information Provided |
|---|---|
| 2.1 | Summary descriptions of command functions |
| 2.2 | Shortest possible command abbreviations |
| 2.3 | Command parameter requirements |
| 2.5 | Keywords suggested for use with each command |

When you enter a password in a request, you must enter a caret (^) before the password as shown in the following example:

```
get testfile pw=^newpass
```

If the response to a request you enter may contain passwords and you have no need to view the passwords, be sure that the keyword **lpw=** is set to **off** (the default). If you display the passwords (**lpw=on**), protect your screen from observation by others.*

* **lpw=off** does not currently work when **set** and **show** are used to list keyword values. This condition should be corrected in the future.

## 9.1 add

Adds a subdirectory to an existing parent directory.

**Shortest command abbreviation: a**

**Parameter(s) required:** *path* . . .

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*`=` | a path for use for the duration of the request |
| `info=` | a text string of up to 80 characters to add comments to the contents of the subdirectory node; string must be enclosed in double quotes (" "); usually used to describe the use to which the subdirectory will be put |
| `pw`*n*`=` | the password to the parent directory; this password becomes the password for the new subdirectory |

**Security considerations:**

To add a subdirectory node, write or insert access to the parent node is required.

**Examples:**

- Add the subdirectory `reports` to your userid root.

  **add** */userid/***reports**

  or

  **add reports**

  To check that `reports` exists, enter

  **list lo=d**

- Add the subdirectories **feb**, **mar**, and **april** to the named root **months**, using the **dir***n*= keyword to simplify the request. Assume that the password **sesame** is required for you to access **months**.

  ```
  add feb mar april dir0=/months pw0=^sesame
  ```

  **sesame** will now also be required for you to access **feb**, **mar**, and **april**.

- Add the subdirectory **procedures** to your userid root and include an explanation of what type of files are to be stored under the subdirectory.

  ```
  add procedures info="files containing \
  information about documentation procedures"
  ```

  To display the explanation, enter

  ```
  list procedures lo=i
  ```

## 9.2 adopt

Replaces the keyword values of the active keyset with the keyword values from a saved keyset. (To save a keyset, use **keep**.)

**Shortest command abbreviation: ado**

**Parameter(s) required:** *keyset* or -

**Keywords and their values:**

| | |
|---|---|
| **kpw=** | the password to the keyset |

**Additional comments:**

If a minus sign (-) is used as the parameter, your userid will be the name of the saved keyset.

Keysets are discussed in Chapter 7.

**Examples:**

- Replace the keyword values of your active keyset with the values from your userid saved keyset.

```
adopt -
```

- Replace the keyword values of your active keyset with the values from the saved keyset named **job33**.

```
adopt job33
```

- Perform the above procedure, but assume that the keyset was saved with the keyset password **tryme**.

```
adopt job33 kpw=^tryme
```

## 9.3 copy

Copies an existing CFS file from one path (*path1*) to a new CFS file on a new path (*path2*).

**Shortest command abbreviation: c**

**Parameter(s) required:** *path1* `to` *path2* ...

**Keywords and their values:**

| | |
|---|---|
| dir*n*= | a path for use for the duration of the request |
| **ngrp=** | an option (**a**, **b**, **n**, or **o**) to determine selection of the storage group for the new copy of the file |
| pw*n*= | the password to the existing source node for *path1* and the password to the parent node for *path2*; for more information, see Chapter 6, Section 6.4 |

**Security considerations:**

To use copy, write or insert access to the destination parent and read access to the existing source file (*path1*) are required. Modify access to the source file is also required if the *user validations* field for the node has one or more entries.

A master user validation entry that was written for *path1* is copied to *path2*. If *path1* inherited its master user validation entry from a higher node, *path2* will inherit its master user validation from a higher node. The entry for *path2* may thus be different from that for *path1*.

An entry (whether for the owner or a non-owner) that appears in the *user validations* field for a file descriptor node is copied along with the file. If the entry contains the access rights modifier **s**, the access conditions for the specified userid will be the same for the copy as for the original file. If access to the original file was determined by an entry at a higher level in the tree, the userid will have no access to the copy unless access is granted by an entry in the new parent or related higher-level node.

To be sure access to the copy of a file is what you intend, examine the validation entries along the path to the new file descriptor node.

**Additional comments:**

If the source file has been assigned to a specific storage group, the copy will be assigned to the same group unless the **ngrp=** keyword is used to change the group. If the source file has not been assigned to a specific storage group, the copy will be stored on the group on which the source file currently resides unless a different arrangement is made using **ngrp=**. For information about storage groups and their use in backing up files, see Chapter 11, Sections 11.6, 11.12, and 11.18.

All the attributes of the source file node, except the storage group and an inherited master user validation entry, are automatically carried over to the new file node. Use the **modify** command if you want to change any of these values (e.g., user validation entries, release date) for the new file node.

After a **copy** is executed, two copies of the file exist. The source file descriptor node has a pointer to the original version; the destination file descriptor node contains a pointer to the new copy.

The **to** in *path1* **to** *path2* is required.

If either *path1* or *path2* is stored in **dir0**, you can use a minus sign (-) to specify that path.

**Examples:**

- Assume that **file1** has as its parent your userid root node and that **dir0** contains the path for your userid root. Make a copy of **file1**, placing the copy under the same node and calling it **file2** (the source and destination parent nodes are the same).

  ```
  copy /userid/file1 to /userid/file2
  ```

  or

  ```
  copy file1 to file2
  ```

  You could also store the path for **file1** in **dir0** and then use a - in the **copy** request.

  ```
  set dir0=/userid/file1
  copy - to /userid/file2
  ```

All the attributes associated with `file1` will also be associated with `file2`.

- Assume that you want to perform the above copy, but that `file1` was saved on storage group A (i.e., the *storage group* field for the file node has been set to `a`) and you want `file2` to be placed on storage group B for backup purposes.

  ```
  copy file1 to file2 ngrp=b
  ```

  or

  ```
  copy file1 to file2 ngrp=o
  ```

  All attributes associated with `file1` except the storage group will also be associated with `file2`.

  Now assume that your userid root contains the user validation entry `010203/r/-/s`. Because `file1` inherits this validation from the userid root, the entry does not appear in the *user validations* field for the node and, therefore, is not copied to the new node for `file2`. However, because `file2` has the same parent, it also inherits the validation. You decide you do not want 010203 to access `file2`.

  ```
  modify file2 aval=010203//-/s
  ```

- Perform the copy shown in the first example, but assume that the file descriptor node for `file1` contains the validation entry `010203/ri/sesame/s` and that you do not want to require userid 010203 to supply a password to access `file2`. First copy the file as shown; the contents of the *user validations* field for `file1` will be copied to the new node.

  ```
  copy file1 to file2
  ```

  Next, enter

  ```
  modify /userid/file2 cval=010203/ri/-/s
  ```

  The *user validations* field for `file2` now contains the entry `010203/ri/-/s`, granting read and insert access with no password requirement to userid 010203.

- Place a copy of the file defined by the path */userid/***test** under the subdirectory (destination parent) that has the path **/programs/runs**. Assume that your userid root includes the validation entry *userid/***rewaibm/reopen/s**, so that the password **reopen** is required to access **test**. The password **reset** is required to access the destination subdirectory.

  ```
  copy /userid/test to 2'test \
  dir2=/programs/runs pw2=^reset pw=^reopen
  ```

  or

  ```
  set dir0=/userid/test pw0=^reopen
  copy 0' to 2'test dir2=/programs/runs \
  pw2=^reset
  ```

  All the file attributes associated with the source version of **test** are associated with the new copy of **test**. However, because the master user validation for the original file **test** was inherited from your userid root, the new copy will inherit its master user validation entry from the new parent. Thus, the password **reset** will be required to access the new copy (i.e., **/programs/runs/test**).

## 9.4 `create`

Creates a userid or named root directory and sets the master user (owner) validation entry for the root.

**Shortest command abbreviation:** **`cr`**

**Parameter(s) required:** *root* . . . or null or -

**Possible keywords and their values:**

| | |
|---|---|
| **`charge=`** | three zeros followed by the five-digit C&TD computer charge number to which the CFS charges associated with files stored under the new node are to be billed |
| **`info=`** | a text string of up to 80 characters to add comments to the contents of the root node; string must be enclosed in double quotes (" "); usually used to describe the use to which the root will be put |
| **`pw`*n*`=`** | the owner password to the root; if specified, the password becomes part of the owner's master user validation entry and will be required of the owner to access the root and any of the nodes created in the tree (unless changes are made later using the **`modify`** command) |

**Additional comments:**

If you use a null or minus sign (-) as the parameter with **`create`**, CFS uses (or attempts to use) the path stored in your default working directory **`dir0`** as the root name. The root name must be unique to the entire CFS.

A leading slash on the root name is optional.

If **`charge=`** is not specified, the default charge number for your UNICOS session will automatically be associated with the node.

There is no way to list the names of your CFS root nodes under **cfsi**. However, all existing CFS nodes are listed in the CFS file **/cfsroots/uroots** (updated daily). You can place a copy of the file in your UNICOS area under the name **uroots** by entering the following command at the **cfsi** prompt:

```
get /cfsroots/uroots
```

The file contains three columns of information. The first column consists of Cray User Numbers (i.e., CFS userids) with any leading zeros removed; your userid appears once for each root node you own followed, in the second column, by the name of the root. All your root names are grouped together. To view those names, use an editor or the **more** command on UNICOS to search the file **uroots** for your userid. Omit a leading zero (e.g., user 010203 would search for 10203). Roots you have created since the last update of **/cfsroots/uroots** will not be listed.

**Examples:**

- Create a root directory using your userid as the root name.

  ```
  create
  ```

  The master user validation entry for this node will be *userid*/**rewaibm/-/s**.

  Verify that the root exists.

  ```
  list lo=d
  ```

- Create a root directory with the name **planning**.

  ```
  create planning
  ```

  Verify that the root exists.

  ```
  list /planning lo=d
  ```

- Create a root node named **energy** to store files related to a project. The computer charge number for the project is 01234, and you want to identify the purpose of the directory.

  ```
  create energy charge=00001234 \
  info="energy: options for conservation"
  ```

- Create a root node named **schedules**. Use your default charge number but specify that you will have to supply the password **whosit** to access the root.

  ```
  create schedules pw=^whosit
  ```

  The master user validation entry for the new root will be *userid*/**rewaibm/whosit/s**. You now decide to give userid 010203 read access to the root with the password **whosit**, so you enter

  ```
  modify /schedules aval=010203/r/^whosit/s
  ```

## 9.5 `delete`

Deletes a file and the associated file descriptor node from CFS.

**Shortest command abbreviation:** `del`

**Parameter(s) required:** *path* . . .

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*`=` | a path for use for the duration of the request |
| `pw`*n*`=` | the password to the path for the file or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

To delete a file, write access to the file is required.

**Additional comments:**

If the path for the file descriptor node is stored in `dir0`, you can use a minus sign (-) as the parameter for the `delete` request. However, after the request is executed, be sure to change the contents of `dir0` to another value before continuing with other requests.

**Examples:**

- Delete the file `reports`, which is stored under your userid root node.

  **delete** */userid/***reports**

  Or, if `dir0` contains the path for your userid root,

  ```
  delete reports
  ```

  You can also use the following command to store the complete path for `reports` in `dir0`

  **set dir0=***/userid/***reports**

  then enter

  ```
  delete -
  ```

- Delete the file `jan` from the root node **`reports`**, which is protected by the password **`enter`**.

```
delete /reports/jan pw=^enter
```

- Delete files **`a`**, **`b`**, and **`c`** from subdirectory **`new`** of the root **`reports`** using a temporary setting for **`dir0`**

```
delete a b c dir0=/reports/new
```

  or using a temporary setting for another directory.

```
delete 5'a 5'b 5'c dir5=/reports/new
```

- Perform the same operation as in the previous example, but assume that the password **`enter`** is required to access the subdirectory **`new`**.

```
delete 5'a 5'b 5'c dir5=/reports/new \
pw5=^enter
```

## 9.6 end

Terminates a `cfsi` session.

**Shortest command abbreviation: en**

**Additional comments:**

Entering

```
end <RETURN>
```

while waiting for a `cfsi` prompt does not terminate your connection with `cfsi` until the outstanding request has been processed. To terminate the connection (and the outstanding request) immediately, enter `<CTRL-c>`.

The `end` command is not required in single-line execution mode (see Chapter 2, Section 2.4).

**Examples:**

- End your `cfsi` session at the `cfsi` prompt.

```
end
```

## 9.7 `free`

Deletes a saved keyset from CFS. (To save a keyset, use **keep**.)

**Shortest command abbreviation:** `fr`

**Parameter(s) required:** *keyset* ... or -

**Keywords and their values:**

| | |
|---|---|
| kpw= | the password to the keyset |

**Additional comments:**

If you use a minus sign (-) as the parameter with **free**, the keyset (if you have created one using **keep**) with your userid as its name is deleted.

Keysets are discussed in Chapter 7.

**Examples:**

- Delete the keyset that was saved using your userid as its name.

```
free -
```

- Delete the saved keysets named `job1` and `job2`.

```
free job1 job2
```

- Delete the keyset **newkey** that was saved with the password **openup**.

```
free newkey kpw=^openup
```

## 9.8 `get`

Transfers a copy of a CFS file to the UNICOS worker system; the CFS file remains intact.

**Shortest command abbreviation: g**

**Parameter(s) required:** *workerfile*:*path* ... or *path* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*= | a path for use for the duration of the request |
| pw*n*= | the password to the path for the file or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

To retrieve a file, read or execute access to the path is required.

**Additional comments:**

If a file with the name specified already exists on the worker system, that file is replaced with the file from CFS.

If `dir0` contains the complete path for the file you want to retrieve, you can use a minus sign (-) to represent *path*.

Specify *workerfile* in the parameter if you want the file to have a different name on UNICOS than it has on CFS; otherwise, the file name will be the same as on CFS (i.e., the last node name in the complete path for the file).

**Examples:**

- Retrieve a copy of the file **procedures** from your userid root node.

  ```
  get /userid/procedures
  ```

  Or, if **dir0** contains the path to your userid root,

  ```
  get procedures
  ```

  You can also enter

  ```
  set dir0=/userid/procedures
  ```

  to place the complete path for **procdures** in **dir0** and then enter

  ```
  get -
  ```

  In each case, the copy on UNICOS will retain the name **procedures**.

- Retrieve a copy of the file **procedures** from your userid root node, but call the UNICOS version **workproc**.

  ```
  get workproc:/userid/procedures
  ```

- Retrieve a copy of the file **a** from the subdirectory defined by the path **/testruns/new/first**, assuming that the password **reopen** is required to access **first**.

  ```
  get dir5=/testruns/new/first pw5=^reopen \
  5'a
  ```

  If the password had been set at the file level rather than at the subdirectory level, the same request would have been required to retrieve the file **a**.

## 9.9 keep

Stores the set of keyword values currently in your active keyset as a saved keyset.

**Shortest command abbreviation:** k

**Parameter(s) required:** *keyset* or -

**Keywords and their values:**

| | |
|---|---|
| kpw= | the password that will be required for access to the keyset |

**Additional comments:**

If you use a minus sign (-) as the parameter with **keep**, your userid will be the name of the saved keyset.

If a keyset with the name specified already exists, it is replaced without warning.

Keysets are discussed in Chapter 7.

**Examples:**

- Store your active keyset, giving it your userid for its name.

```
keep -
```

- Store your active keyset, giving it the name **keyseta**.

```
keep keyseta
```

- Store your active keyset, giving it the name **keyseta**, specifying the password **valset** for access.

```
keep keyseta kpw=^valset
```

## 9.10 `list`

Lists information (i.e., attributes; the content of fields) from directory and file descriptor nodes. (To list keyword values, use **set** and **show**. See Chapter 9, Section 9.4 for information about how to list roots.)

**Shortest command abbreviation:** `l`

**Parameter(s) required:** *path* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*= | a path for use for the duration of the request |
| `lo=` | one or more of the list options **a, d, g, i, s,** and **u**; used to specify what information you would like to view |
| `lpw=` | either `on` or `off`, depending on whether you want any passwords to be displayed; should be set to `off` (the default) unless there is a need to view passwords |
| `pw`*n*= | the password to the path for the node or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

Read access to the node is required for most list options; modify access is required to list user validation entries.

**Additional comments:**

You can use a blank as the parameter in a list request; CFS will automatically use the path stored in `dir0`. Using a minus sign (-) has the same effect.

The output from `list` varies, depending on the value assigned to the list option (`lo=`) keyword and the type of node being listed (see `lo=` in Chapter 11).

— If the path specifies a directory node, the default value for `lo=` is `d`; the names and types of the node's immediate descendants are returned.

— If the path specifies a file descriptor node, the default value for `lo=` is `g`; general information for that file is returned.

Specify multiple list options in any order with no delimiters as in the following example:

```
list lo=gu
```

**Examples:**

- List information from the node named in your default working directory using the default value for `lo=` for that type of node.

  ```
  list
  ```

  or

  ```
  list -
  ```

- List the validation entries for root `a`, which has the password `enter`.

  ```
  list lo=u /a pw=^enter
  ```

  Because the default value for `lpw=` is used (i.e., `off`), any passwords in the validation entries are displayed as strings of percent signs.

- List general information and user validations, including passwords, for files `a` and `b`, which are stored under subdirectory `talk` of the root `trip`.

  ```
  list lo=ug dir0=/trip/talk a b lpw=on
  ```

  Be sure to protect your screen while the passwords are displayed and to clear it as soon as possible.

## 9.11 `modify`

Changes information in directory and file descriptor node fields (i.e., changes specific node attributes).

**Shortest command abbreviation:** `mod`

**Parameter(s) required:** *path* ...

**Keywords and their values:** The first two keywords may be used to specify the path for the node being modified and the password, if any, required for access to that node. Use the remaining keywords to change the contents of various fields for the specified node.

| | |
|---|---|
| **dir*n*=** | a path for use for the duration of the request |
| **pw*n*=** | the password to the path for the node or to the path contained in the corresponding dir*n*, depending on the form of the request |
| **aval=** | user validation entry to be added for a node or initial change in the master user validation for a node; format is *userid/rights/password/modifier*, where *userid* is the userid of the user with whom the rights are to be associated |
| **cval=** | validation entry to replace an existing entry in the *user validations* field for a node; format is *userid/rights/password/modifier*, where *userid* is the userid of the owner or other user with whom the rights are to be associated |
| **dval=** | the userid associated with a user validation to be deleted |
| **ncharge=** | a new computer charge number for a root node (prefixed by three zeros) |
| **ninfo=** | a new text string of up to 80 characters to replace the current contents of the *info* field; string must be enclosed in double quotes (" ") |
| **nname=** | a new node name; do not attempt to change the name of another user's root |

| | |
|---|---|
| `nrel=` | a new file release date |
| `nuse=` | a new estimate for file activity |
| `setgrp=` | **on** to lock in the storage group for a file if a storage group was not specified when the file was saved. |

**Security considerations:**

The following access rights are required for the changes possible with `modify`.

| Keyword | Type of Node | Required Rights |
|---|---|---|
| `aval=` | All | Modify or Bestow |
| `cval=` | All | Modify |
| `dval=` | All | Modify |
| `ncharge=` | Root | Modify |
| `ninfo=` | All | Modify or Write |
| `nname=` | All | Modify or Write |
| `nrel=` | File descriptor | Modify or Write |
| `nuse=` | File descriptor | Modify |
| `setgrp=` | File descriptor | Modify |

**Additional comments:**

If *path* is stored in dir0, you can use a minus sign (-) as the parameter in a `modify` request.

**Examples:** (see also individual keyword descriptions)

- Add a validation entry for user 010203 to your userid root node.

  **modify** */userid* **aval=010203/r/-/s**

  Or, if **dir0** contains the path for your userid root,

  ```
  modify - aval=010203/r/-/s
  ```

- Assume that the validation entry **010203/r/enter/s** is associated with the subdirectory defined by the path **/reports/monthly** and that the password **enter** is also required for you to access **monthly**. Delete the validation entry for user 010203.

  ```
  modify /reports/monthly pw=^enter \
  dval=010203
  ```

- Assume that your root **reports** contains the master validation entry */userid*/**rewaibm/enter/s** and that its subdirectory **monthly** has inherited that entry. Change the password for your access to **monthly**.

  ```
  modify /reports/monthly pw=^enter \
  aval=userid/rewaibm/^newpass/s
  ```

- Assume that your root **reports** contains the master validation entry */userid*/**rewaibm/-/s** and that the password **enter** has been set for your access to its subdirectory **monthly** (i.e., the *master user validation* field contains the entry */userid*/**rewaibm/enter/s** and the *user validations* field also contains that entry). Change the password for your access to **monthly**.

  ```
  modify /reports/monthly pw=^enter \
  cval=userid/rewaibm/^newpass/s
  ```

- Assume that you want to set a release date of June 1, 1990 for the file **results**, which is a direct descendant of your userid root.

  ```
  modify /userid/results nrel=90/06/01
  ```

## 9.12 `move`

Moves an existing node (*path1*) and its subtree, if any, to a new node (*path2*).

**Shortest command abbreviation:** `mov`

**Parameter(s) required:** *path1* `to` *path2* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*`=` | a path for use for the duration of the request |
| pw*n*= | the password to the source node for *path1* and to the new parent node for *path2* |

**Security considerations:**

To accomplish a `move` request, modify access to the source node *path1* and write or insert access to the destination parent node are required.

An entry (whether for the owner or a non-owner) that appears in the *user validations* field for a source node is moved along with the node.

When you move a node belonging to another user to a node belonging to you or to a new root node, you become the owner of the moved node. Conversely, other users can use `move` to gain possession of your nodes. For this reason, it is good practice not to grant other users modify access to your root nodes and to grant them modify access to other nodes only when special circumstances require.

When *path1* and *path2* belong to the same user:

- A master user validation entry that was written for the source node is moved along with the node. However, if *path2* is a new root, CFS will modify the entry to give the owner full access rights.
- If *path1* inherited its master user validation entry from a higher node, *path2* will do the same. The entry for *path2* may thus be different from that for *path1*.

To be sure access to a moved node is what you intend, examine the validation entries along the path to the node at its new position.

**Additional comments:**

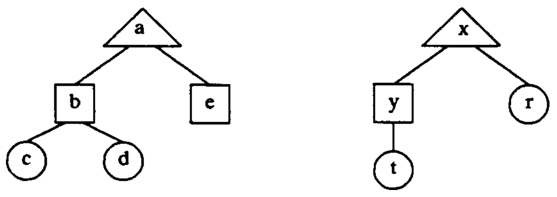
If either *path1* or *path2* is stored in `dir0`, you can use a minus sign (`-`) to represent that path in a `move` request.

Only the nodes in the CFS tree are manipulated by **move**. The actual files represented by the file descriptor nodes are not copied or relocated.

If you use **move** to create a new root, the new root is created using your default charge number. If the new root name is a userid, it must be your userid.

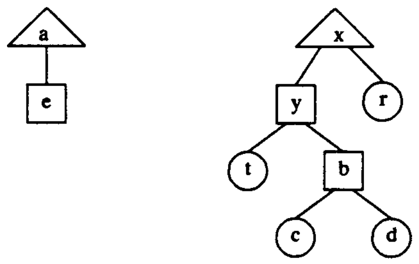
**Examples:** In the following examples, diagrams of the original tree structure(s) precede the request; the tree structure(s) resulting from the request follow.

- Example 1:

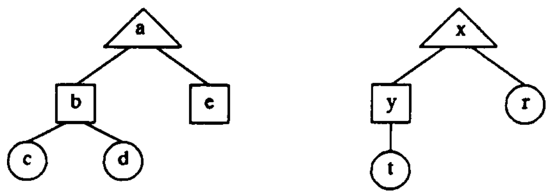


Place node **b** (and its subtree) of root **a** under node y of root **x**. y is the destination parent node.

```
move /a/b to /x/y/b
```

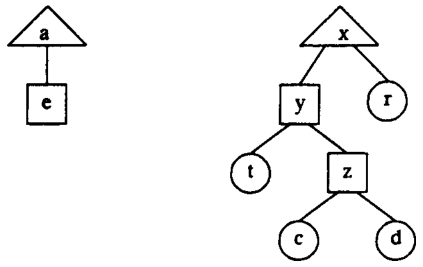


- Example 2:



Place node b (and its subtree) of root a under node y of root x, but change the name of node b to z in its new location.

```
move /a/b to /x/y/z
```

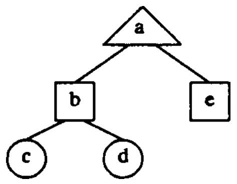


- Example 3:

Perform the preceding move, but assume that the password **letmein** is required to access node b and the password **sesame** to access node y.

```
set dir1=/a/b pw1=^letmein
move 1' to 2'z dir2=/x/y pw2=^sesame
```

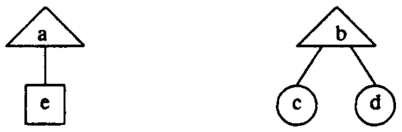
- Example 4:



Create a new root node b from the subdirectory b (and its subtree) of root a. Assume that the master user validation entry for root a is *userid*/**rewaibm/letmein/s** and that node b inherits this entry.

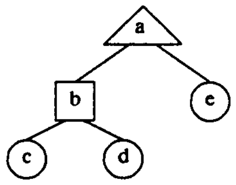
```
move 1' to /b dir1=/a/b pw1=^letmein
```

The master validation for the new root node **b** will be *userid*/**rewaibm/letmein/s**.



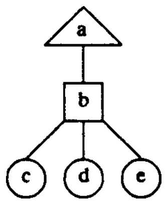
If you had written the master user validation entry *userid*/**reaibm/-/s** for the subdirectory node **b**, the master user validation entry for the new root would have been *userid*/**rewaibm/-/s**.

- Example 5:

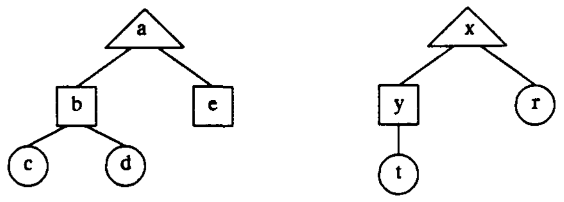


Change the parent directory of file **e** from root **a** to subdirectory **b** of root **a**.

```
move /a/e to /a/b/e
```



- Example 6:



Make root **a** and its associated tree a part of the tree associated with root x by making **a** a direct descendant of x. Assume that root **a** has the master user validation entry *userid*/**rewaibm/sesame/s** and that root x has the master user validation entry *userid*/**rewaibm/reopen/s** and the user validation entry 010203/**w/reopen/s**. First enter

```
set dir0=/a pw=^sesame
```

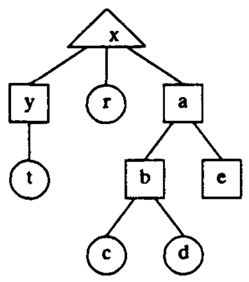
Then enter either

```
move 0' to 1'a dir1=/x pw1=^reopen
```

or

```
move - to 1'a dir1=/x pw1=^reopen
```

Both validation entries associated with root **a** will be moved with the node.



## 9.13 `remove`

Removes an *empty* directory node (root or subdirectory) from CFS.

**Shortest command abbreviation: `rem`**

**Parameter(s) required:** *path* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*`=` | a path for use for the duration of the request |
| `pw`*n*`=` | the password to the path for the directory or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

To delete a directory node, write access to the path is required.

**Additional comments:**

If the directory node has descendants, the request will fail.

If the path for the directory node is stored in `dir0`, you can use a minus sign (-) as the parameter for the **`remove`** request. However, after the request is executed, be sure to change the contents of `dir0` to another value before continuing with other requests.

**Examples:**

- Remove the subdirectory **`alternate`** from the root named **`plans`**.

  ```
  remove /plans/alternate
  ```

- Assuming that the root **`plans`** now has no descendants, remove **`plans`**.

  ```
  remove /plans
  ```

- Remove the subdirectories **`a`**, **`b`**, and **`c`** from the parent subdirectory **`options`**, which is a direct descendant of your userid root. The password **`codes`** is required to access **`options`**.

  ```
  remove a b c dir0=/userid/options pw=^codes
  ```

- Remove the directory node named in the working directory `dir4`.

```
remove 4'
```

## 9.14 `replace`

Replaces an existing CFS file with a file from the UNICOS worker system.

**Shortest command abbreviation:** `r`

**Parameter(s) required:** *path* ... or *workerfile*:*path* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*`=` | a path for use for the duration of the request |
| `pw`*n*`=` | the password to the path for the CFS file or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

To replace a file, write access to the node is required.

**Additional comments:**

The workerfile name is required as part of the parameter only if the file is to have a different name on CFS.

All the attributes (e.g., user validation entries, password, storage group) associated with the file descriptor node before the request is executed remain the same after the replacement. Use the `modify` command to make changes to these attributes.

If the path defining the file to be replaced is stored in `dir0`, you can use a minus sign (-) to represent *path* in the `replace` request.

The original workerfile remains intact after the `replace` request has been executed; delete the workerfile if there is no reason to retain it.

**Examples:**

- Replace **test1**, which is a direct descendant of your userid root node, with the UNICOS file called **test1**.

  ```
  replace /userid/test1
  ```

  Or, if **dir0** contains the path for your userid root,

  ```
  replace test1
  ```

  You might, instead, use the following two requests, particularly if you intend to use the file as the parameter in several requests:

  ```
  set dir0=/userid/test1
  replace -
  ```

- Replace the file **choices**, which is a direct descendant of your userid root node, with the UNICOS file called **selection**.

  ```
  replace selection:/userid/choices
  ```

- Replace the CFS file defined by the path */userid/***reports/monthly/jan** with the UNICOS file called **jan**. Assume that the path is protected by the password **stars**.

  ```
  replace /userid/reports/monthly/jan \
  pw=^stars
  ```

## 9.15 `save`

Creates a new CFS file (and associated file descriptor node) by copying a file from the UNICOS worker system to CFS.

**Shortest command abbreviation: `s`**

**Parameter(s) required:** *path* ... or *workerfile*:*path* ...

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*= | a path for use for the duration of the request |
| grp= | an option (a, b, or -) to determine selection of the storage group for the new file |
| pw*n*= | the password to the path for the CFS file or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

To save a file on CFS, write or insert access to the parent node is required.

The new file node on CFS inherits the access of its parent node. Check the validation entries along the path leading to the file node to be sure the access is what you intend.

**Additional comments:**

If a file of the same name exists in the CFS directory specified, a warning message is sent; you are not permitted to overwrite the existing file.

The workerfile name is required only if the file is to have a different name on CFS.

The original workerfile remains intact after the **save** request has been executed; delete the workerfile if there is no reason to retain it.

The new file descriptor node will assume the relevant attributes defined in your active keyset. Use the `modify` command to change the attributes (e.g., release date) associated with the new file descriptor node and to change access rights inherited from the parent node.

If the path for the new file descriptor node is stored in `dir0`, you can use a minus sign (-) to represent *path*.

**Examples:**

- Save the file **`test1`** as **`test1`** under your userid root.

  ```
  save /userid/test1
  ```

  Or, if **`dir0`** contains the path for your userid root,

  ```
  save test1
  ```

- Save the UNICOS file **`test1`** as **`function`** under your userid root, assuming that the password **`reopen`** is required to access the root.

  ```
  save test1:/userid/function pw=^reopen
  ```

- Save the UNICOS file **`jan`** on CFS as **`jan`** under the subdirectory defined by the path */userid/***`reports/monthly`**. The password **`letmein`** is required to access **`monthly`**.

  ```
  set dir0=/userid/reports/monthly pw0=^letmein
  save jan
  ```

- Save files **`a`**, **`b`**, and **`c`** under subdirectory **`options`** of root **`tests`**. Specify that all three files are to be placed on storage group B.

  ```
  save a b c dir0=/tests/options grp=b
  ```

## 9.16 **set**

Lists or changes values for keywords in your active keyset. (Use **list** to display node attributes and **show** to list the values of keywords in saved keysets.)

**Shortest command abbreviation: se**

**Keywords and their values:**

Use one of the following requests to list your active keyset:

```
set        or        set -
```

You can use the keyword **lpw=*** in either request, but note that doing so makes a permanent, rather than temporary, change in the value of the keyword in your active keyset.

To list the value of an individual keyword in your active keyset, specify the keyword in standalone form (i.e., without an equal sign and value) in a **set** request as follows:

**set** *keyword*

You may use any keyword in this way.

To change the value of a keyword in your active keyset, use **set** as follows:

**set** *keyword=value*

The new value will remain in effect until changed with another **set** request or until you end your **cfsi** session. In this manual, **set** is used to change the value of only the following three keywords:

| | |
|---|---|
| **dir***n*= | a path |
| **lpw=** | **on** or **off** to indicate whether user-defined passwords will be displayed when they occur in a **list**, **set**, or **show** response;** **off**, the default, should be used unless there is a reason to view the passwords (see discussion and examples in Chapter 11, Section 11.10) |

* Although you can use **lpw=** with **set** when listing keyword values, **lpw=off** does not currently work, so any passwords contained in keyword values are always displayed.

** **lpw=off** does not currently affect the display of passwords when **set** and **show** are used to display keyword values. A correction to this problem is expected.

| | |
|---|---|
| pw*n*= | the password to the path contained in the corresponding **dir*n*** |

When a non-default value for another keyword is desired, that value is set for the duration of a request by including the keyword and value in the request. For example,

```
save file1 grp=a
```

**Additional comments:**

Forgetting that you have changed the value of a keyword in your active keyset can cause problems. If you begin receiving error responses you do not anticipate, check the contents of your working directories and any working directory passwords you have set to be sure they are what you currently need.

If a request contains no command, the **set** command is assumed. Thus,

```
set dir2=/userid/reports
```

is the same as

```
dir2=/userid/reports
```

If you use a minus sign (-) as the keyword value in a **set** request, the value of the keyword in the active keyset becomes the system default (see Table 2.4). Thus,

```
set lpw=-
```

sets the value of **lpw=** to **off**.

**Examples:**

- List all your active keyset values.

  ```
  set
  ```

- Determine the value that is in effect in your active keyset for the **dir2=** keyword.

  ```
  set dir2
  ```

- Reset the current value of working directory **dir3** to the system default (an empty path).

  ```
  set dir3=-
  ```

- Assign the path */userid/*`project/reports` to `dir3` and set `pw3=` to `comein`, the password required to access the path.

  ```
  set dir3=/userid/project/reports pw3=^comein
  ```

  You can now easily save files under `reports` using the `3'` notation as in

  ```
  save 3'file1
  ```

  Later in your session you enter

  ```
  modify - aval/userid/rewaibm/-/s
  ```

  to remove the password requirement for access to `reports`. You then remember one more file you want to save and enter

  ```
  save 3'testfile
  ```

  CFS returns the error response `no access to the node` because `pw3` continues to supply a password even though none is now required. The request will succeed if you first enter

  ```
  set pw3=-
  ```

  to return `pw3=` to its null default value.

- Assume that you want to save several files (`test1`, `test2`, and `test3`) under the subdirectory defined by the path */userid/*`trials` and that you want the files placed on storage group A.

  ```
  set dir0=/userid/trials
  save test1 test2 test3 grp=a
  ```

## 9.17 **show**

Displays contents, date of creation, and date last saved for one or more of your saved keysets. (To display the values of your active keyset, use **set**. To save a keyset, use **keep**.)

**Shortest command abbreviation: sh**

**Parameter(s) required:** *keyset* . . . or blank

**Keywords and their values:**

| | |
|---|---|
| **kpw=** | the password required to access the keyset for which information is being requested |
| **lpw=** | **on** or **off** depending on whether you want any user-defined passwords included in a keyset (e.g., **pw3=letmein**) to be displayed;* **off**, the default, should be used unless there is a need to view passwords |
| **so=** | **o**, **n**, or **a** to indicate what information you want to have displayed. **o** displays a specific saved keyset; **a** displays all your saved keysets; **n** displays the names of all your saved keysets. **n** is the default.** |

**Security considerations:**

If a keyset is password-protected and the password is not supplied using the **kpw=** keyword, the keyset will not be displayed.

You are the only user who can display your saved keysets.

**Additional comments:**

Do not use a parameter with **n** or **a**. Supply a parameter with **so=o**; with this keyword and option, a blank parameter represents your userid saved keyset.

* The **off** setting for this keyword does not currently mask passwords in a **show** response, but should in the future.

** Currently, you must include **so=o** in every **show** request to get consistent responses. Implementation of the **n** and **a** options is expected in the future.

**Examples:**

- Display all the contents of your userid saved keyset. Assume the keyset is not password-protected.

  ```
  show so=o
  ```

  or

  show *userid* so=o

- Perform the above procedure, but assume the keyset is protected by the password `dosay`.

  ```
  show so=o kpw=^dosay
  ```

- Display the entire contents, including pw*n*= values, of the saved keysets named `first` and `second`. The keysets are not password-protected.

  ```
  show so=o first second lpw=on
  ```

## 9.18 `status`

Checks the status of CFS and its devices and the status of any of your requests not yet completed.

**Shortest command abbreviation:** `sta`

**Additional comments:**

CFS automatically sends periodic status messages when a request requires more than about a minute to execute (e.g., you may get a `waiting for storage device` message if the file you have specified in the request must be retrieved from tape). If you are running `cfsi` in the background, you may want to direct these messages to a file to avoid having them displayed on your screen.

If you enter a `status` request, the information you receive includes

- the number of CFS requests in the queue waiting to be executed;
- the number of CFS requests being executed; and
- the status of disk and tape storage for storage groups A and B.

If you have one or more requests in the queue or being executed, the response will include the following, as applicable:

- the command name,
- when the request was sent to CFS, and
  - — where the request is in the queue,
  - — what unavailable device (if any) is inhibiting execution of the request, or
  - — when execution started and what percentage of the current file transmission has been completed.

If you enter `status` while waiting for a response to a request, the `status` response will not be returned until after the original request has been executed. You will, however, continue to receive the periodic system messages.

Enter `<CTRL-c>` to exit `cfsi` while waiting for a request to execute. The request will terminate.

If, when using **status**, you do not receive a `cfsi` or UNICOS system prompt when expected, try entering <RETURN>.

**Examples:** (Sample responses are shown following the requests.)

- Display status information when you do not have a request pending.

  ```
  status
  ```

  ```
  queued requests:  0        active requests:  1
  status for:      disk         mass storage
  groupa            up               up
  groupb            up               up
  ```

- You want to retrieve the file `file1` from tape using single-line execution in the background but do not want to submit the job unless CFS is running. At the UNICOS prompt you enter

  ```
  cfsi status
  ```

  You receive a response similar to that above indicating that CFS is up, so you enter

  ```
  nohup cfsi get file1 &
  ```

  Using the **nohup** command causes any periodic status messages from the system to be directed, as part of the output, to the file **nohup.out**.

  If you have a batch job that invokes `cfsi` to perform a function such as saving output files, you may want to use a **status** request early in the job to determine whether to continue or not.

- You decide to check on the status of the **get** request submitted in the previous example. You can invoke **cfsi** and then enter

  ```
  status
  ```

  or simply enter

  ```
  cfsi status
  ```

  at the UNICOS prompt.

```
queued requests:  0        active requests:  4
status for:    disk          mass storage
groupa          up              up
groupb          up              up
request status:
command arrived          queue waiting on       % of file
started   .    .    .    .    .    .    .    .    .    transmitted
get      13:36       13:36 00000       queue     61
```

## 9.19 **`store`**

Saves or replaces files on CFS.

**Shortest command abbreviation: `st`**

**Parameter(s) required:** *path* . . . or *workerfile*:*path* . . .

**Keywords and their values:**

| | |
|---|---|
| `dir`*n*= | a path for use for the duration of the request |
| **grp=** | an option (`a`, `b`, or `-`) to determine selection of the storage group for a new file (applies only if a **`save`** is executed) |
| **pw***n*= | the password to the path for the CFS file or to the path contained in the corresponding `dir`*n*, depending on the form of the request |

**Security considerations:**

Write access to the path is required if the **`store`** replaces an existing file; if a file with the name indicated in *path* does not exist (i.e., the **`store`** executes a **`save`**), write or insert access to the parent node is required.

**Additional comments:**

Before **`store`** is executed, the active keyset is checked to see if any pertinent keywords are set. If a file with the specified path does not exist, **`store`** executes a **`save`** request using the keywords relevant to **`save`**. If the file does exist, **`store`** executes a **`replace`** using the keywords relevant to **`replace`**.

An existing CFS file is overwritten without warning, so use **`store`** carefully.

The workerfile name is required in the parameter only if the file is to have a different name on CFS.

If *path* is stored in `dir0`, you can use a minus sign (-) to represent the path in the **`store`** request.

The original workerfile remains intact after the **`store`** request has been executed; delete the workerfile if there is no reason to retain it.

Examples:

- Save or replace the file named **test1** under your userid root.

  ```
  store /userid/test1
  ```

  Or, if **dir0** contains the path for your userid root,

  ```
  store test1
  ```

- Store the UNICOS file named **test1** as **test1sub** under the subdirectory defined by the path */userid/***experiments/firstrun**. The password `letmein` is required to access `firstrun`.

  ```
  store test1:test1sub pw=^letmein \
  dir0=/userid/experiments/firstrun
  ```

- Store the file named **test1** under your userid root and specify that the file be located on storage group B.

  ```
  store test1 grp=b
  ```

  Note that the **grp=** keyword is used if a **save** is executed, but ignored if a **replace** occurs (i.e., if a file named **test1** already exists under your userid root). If a **replace** occurs, the contents of the *group* field for the node remain unchanged.

# 10. Parameter Descriptions

A parameter is used in a request to specify the object upon which the associated command is to be performed. There are four types of parameters: keysets, paths, roots, and workerfiles. They are described in Sections 10.1 through 10.4.

Each command has specific parameter requirements as described in Chapter 9 and summarized in Table 2.3. With some commands, a minus sign (-) or a null (i.e., a blank) may be used to represent a parameter. For details, see the individual command descriptions in Chapter 9.

Multiple parameters are permitted with many commands. When you use multiple parameters, CFS interprets the request as multiple requests, one for each parameter, in the order given. For example, the request

```
save file1 file2 dir0=/userid/collect
```

which has the two parameters `file1` and `file2`, causes CFS to execute

```
save file1 dir0=/userid/collect
save file2 dir0=/userid/collect
```

The parameter descriptions in Sections 10.1 through 10.4 follow the format outlined below:

- **Parameter name.**
- **Brief description.**
- **Commands used with:** A list of the commands covered in this manual that require or can use the parameter.
- **Naming restrictions.**
- **Comments and examples:** Comments about use of the parameter and examples of its use. The examples, in all cases, assume that no passwords are required.

## 10.1 Keyset Parameter

The name of a saved keyset.

**Commands requiring a keyset parameter:**

**`adopt, free, keep, show`**

**Naming restrictions:**

A keyset name must be 1 to 16 characters in length. Legal characters for a keyset name are

`a-z, A-Z, 0-9, $, %, *, +, -, .,` and `_`.

**Comments and examples:**

A keyset is a set of all keywords and their assigned values. A saved keyset is a non-active keyset that is being held for future use; it is preserved by CFS from session to session until you delete it or until it is automatically deleted after a year of disuse. For a discussion of keysets, see Chapter 7.

Other users may not use your keysets.

You can specify a keyset password when you save a keyset. The password will then be required for future access to the keyset. For information about the use of keyset passwords, see Chapter 7, Section 7.2 and Chapter 8, Section 8.3.

When used with the **`keep`** command, the keyset parameter is the name you are giving the keyset you are storing. Thus, if you want to store your active keyset, giving it the name **`keyset2`**, enter

```
keep keyset2
```

When used with **`adopt`**, **`free`**, or **`show`**, the keyset parameter is the name of an existing saved keyset. For example, you can make **`keyset2`**, saved above, your active keyset by entering

```
adopt keyset2
```

You can use a minus sign (-) as the keyset parameter; CFS will substitute your userid as the keyset name. For example,

```
keep -
```

stores your active keyset as the keyset named *userid*, and

```
adopt -
```

retrieves that keyset and makes it the active keyset.

If you omit the keyset parameter in a `show` request, CFS will use your userid keyset.

## 10.2 Path Parameter

A sequence of node names that identifies a node.

**Commands requiring a path parameter:**

`add, copy, delete, get, list, modify, move, remove, replace, save, store`

**Naming restrictions:**

The node names in a path must be separated by slashes. Each node name can be up to 16 characters in length. Legal characters for a node name are

`a-z, A-Z, 0-9, $, %, *, +, -, .,` and `_`.

The maximum length of a path, including slashes, is 96 characters. A path can be no more than 48 levels deep; 6 or 7 levels is a practical limit.

**Comments and examples:**

Paths and tree structures are described in Chapter 3. The node defined by a path may be password-protected. For information about password requirements and the use of passwords, see Chapter 8, Section 8.1.4.

A complete path in a CFS tree is defined as the name of each node from the root downward to the specified node, where each node is chained together and preceded by a slash (/). The sequence describes to CFS the route to follow through a tree to reach the specific node. For example, the complete path for the file descriptor node **prgm1** in Figure 10.1 is */userid/***prgm1**,

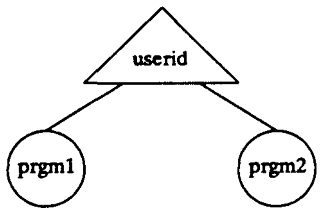


**Fig. 10.1.**

and the complete path for the file descriptor node `jan` in Figure 10.2 is */userid/*`reports/monthly/jan`.

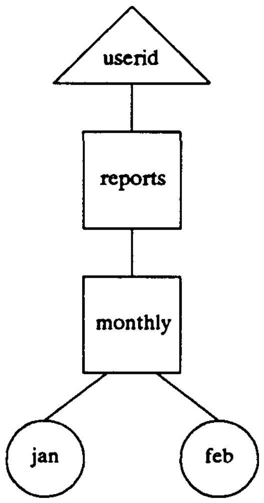


**Fig. 10.2.**

To execute a request requiring a path parameter, CFS must be able to determine the complete path using information from the request and from your working directories.* The following general rules apply:

- If the first character of a path is a slash, the path is considered to be a complete path. Thus, you can specify the complete path for the node `jan` in Figure 10.2 by entering */userid/*`reports/monthly/jan` as the parameter in a request.
- If the first character of a path is not a slash, as in `reports/monthly/jan`, CFS appends the contents of a working directory to form a complete path. You can indicate the appropriate working directory in one of two ways:

* Working directories are introduced in Chapter 3, Section 3.2 and discussed in detail in Chapter 6.

— You can omit reference to a working directory in the request, causing CFS to append the contents of working directory `dir0`. This directory, by default, contains the path for your userid root. Thus, if you are working with the tree in Figure 10.1, have not used the `set` command to change the contents of `dir0`, and enter the request

```
get prgm1
```

CFS appends */userid* to the parameter specification and executes the request

```
get /userid/prgm1
```

— You can use the *n'* notation to specify the directory, where *n* is the number of the working directory (0 through 9). For example, if `dir6` contains */userid/*`reports/monthly` and you enter

```
get 6'jan
```

CFS executes

```
get /userid/reports/monthly/jan
```

If the *n'* notation is used alone, CFS takes the complete path from the specified `dir`*n*. Thus, if `dir6` in the above example had contained */userid/*`reports/monthly/jan`, you could have entered

```
get 6'
```

to retrieve the file `jan`.

- If you use a minus sign (-) as the parameter, CFS takes the complete path from your default working directory. Thus, if `dir0` contains */userid/*`reports` and you enter

```
get -
```

CFS executes

```
get /userid/reports
```

to retreive a copy of the file `reports`.

- If you use a null path (only allowed with **list**), CFS takes the complete path from your default working directory. Thus, if **dir0** contains the default value */userid* and you enter

  ```
  list
  ```

  CFS lists information about your userid root.

> If you begin receiving the response **node does not exist**, particularly when making use of **dir0**, check the contents of the working directory you are using to be sure the stored path is the one that is currently needed.

If you specify a path that does not exist or make a syntax error in the path specification, CFS will not execute your request and will return an error response. If you enter a path incorrectly but, in doing so, describe an existing path, the request will be executed with results you had not anticipated. For example, assume you have root nodes named */userid* and **/reports** and these roots have the associated trees shown in Figure 10.3.

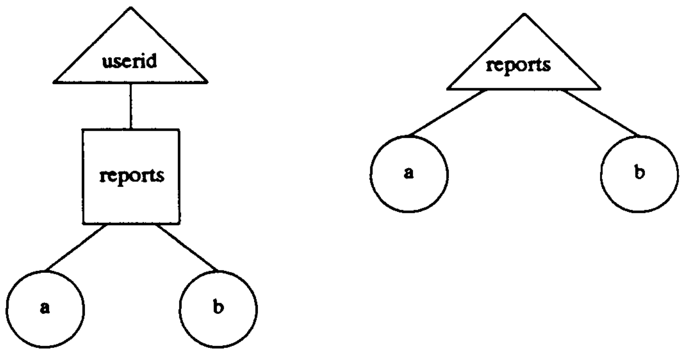


**Fig. 10.3.**

Also assume that you want to retrieve the file **a** that is stored as a third-level descendant of your userid root. You intend to make use of **dir0** to simplify the request and enter

```
get /reports/a
```

Because you preceded the parameter with a slash, however, CFS takes the path to be a complete path, does not append */userid*, and retrieves the file **a** that is a descendant of the root **reports**.

The **copy** and **move** commands require that both a source and a destination path be specified. The format is *path1* **to** *path2*. For more information, see the command descriptions for **copy** and **move** in Chapter 9.

> If your request involves a UNICOS file that is or will be located in a directory other than the one from which you invoked **cfsi** or if a file you specify in a request will have different names on UNICOS and CFS, you must use a workerfile parameter with the path parameter. For information, see Section 10.4.

## 10.3 Root Parameter

The name (or path) of a userid or named root node.

**Commands requiring the root parameter: `create`**

**Naming restrictions:**

There are two types of root names.

- A userid root uses as its name a personal identifier specifically reserved for a user. Your personal identifier is your six-digit Cray User Number (see the Preface).
- A named root can be created using the name of your choice. The root name, however, must start with an alphabetic character, be no longer than 16 characters in length, and incorporate only the characters

  `a-z, A-Z, 0-9, $, %, *, +, -, .,` and `_`.

  The root name you select must be unique to CFS. If a root with the name you select already exists, you will receive an error response and will have to try another name.

**Comments and examples:**

You must create at least one root node before you can store any files on CFS. A root node is the origin of a tree structure, which is the basis for file organization on CFS. Root nodes and tree structures are discussed in detail in Chapter 3. Roots may be password-protected. For information about password requirements and the use of passwords, see Chapter 8, Section 8.2.4.

When you create a root node, you are the owner of that node and all its descendants (i.e., the master user validation entry contains your userid).

See Chapter 9, Section 9.4 for information about how to list the names of all your root nodes.

You may specify a root parameter by name or by its path. Thus,

```
create reports
```

and

```
create /reports
```

are equivalent.

If you use a null or a minus sign (-) as the parameter with **create**, CFS will use the path (minus the leading slash) stored in the default working directory **dir0** as the root name. Thus, if **dir0** contains the default value */userid*, you can enter

```
create
```

or

```
create -
```

to create your userid root. If **dir0** contains a path with more than one level, CFS will return an error response because a root name cannot contain the slash character (/).

## 10.4 Workerfile Parameter

The name of a file on UNICOS (i.e., the worker system); used in combination with a CFS path using the format *workerfile*:*path* when different names for a file apply on UNICOS and CFS or when you are referring to a UNICOS file that does not reside in the directory from which you invoked `cfsi`.

**Commands that can use a workerfile parameter:**

`get, replace, save, store`

**Naming restrictions:**

Workerfile names must conform to UNICOS naming conventions.

If the workerfile is associated with your working directory on UNICOS (i.e., the directory from which you invoked `cfsi`), use only the file name. If the workerfile is associated with another UNICOS directory, specify the UNICOS path for the file as described in Appendix C.* For simplicity, the examples in this manual make two assumptions:

- when entering a `get` request, you have invoked `cfsi` from the UNICOS directory into which you want the file placed; and
- when entering a `save`, `store`, or `replace` request, you have invoked `cfsi` from the UNICOS directory containing the file to be saved.

For example, if you have a subdirectory called `subdir` under your login directory on UNICOS and the file `testrun` is located in that subdirectory, you could invoke `cfsi` from your login directory and then enter

```
save subdir/testrun:testrun
```

to save a copy of `testrun` on CFS. However, this manual assumes that you enter

```
cd subdir
```

* Appendix C also describes how to use `cd` to change UNICOS directories from within `cfsi`.

on UNICOS, and then invoke `cfsi` from `subdir`, so that the simple request

```
save testrun
```

is all that is required to save `testrun`.

**Comments and examples:**

The *workerfile*: portion of *workerfile:path* is optional as described below.

- If you want the file on CFS and the corresponding file on UNICOS to have the same name, omit *workerfile*:. CFS will use the last node name in the CFS path (i.e., the file name on CFS) as the file name on UNICOS. For example, the request

  ```
  get /userid/testrun
  ```

  gets a copy of the CFS file named `testrun` and places it in the UNICOS directory from which you invoked `cfsi`. The new file on UNICOS is also named `testrun`.

- If you want the file on CFS and the corresponding file on UNICOS to have different names, include *workerfile*:. For example, if you have a file named `testrun` on UNICOS and want to save it on CFS directly under your userid root as `run1`, enter

  ```
  save testrun:/userid/run1
  ```

  or, if `dir0` contains the path for your userid root,

  ```
  save testrun:run1
  ```

You cannot store a *workerfile:path* specification in a working directory.

You may use a space on either side of the colon as shown below.

```
replace awards2 : awards
```

The workerfile name must always appear to the left of the colon.

# 11. Keyword Descriptions

Keywords are used to exercise options permitted with a command, to define node attributes (i.e., the content of node fields; see Chapter 3, Section 3.1 and Appendix A), or to define conditions for a request. They are literal names to which values may be assigned. The values are sometimes literal (shown in monospaced type in the descriptions) and sometimes symbolic (shown in italics). The format for defining a keyword is

**keyword=***value*

A keyset is a complete set of keywords and their values. An active keyset is the keyset currently in use; a saved keyset is one that has been stored for future use. Keysets are described in Chapter 7.

When you invoke **cfsi**, an active keyset is created with each keyword set to its default value. Each command has certain keywords that are associated with it. When you enter a request, CFS automatically gets the values for any of those keywords that are not defined in the request from your active keyset.

You change the value of any keyword in your active keyset by using **keyword=***value* in a **set** request, where *value* is a new value. However, this manual does this only with **dir***n***=**, **pw***n***=**, and **lpw=**. You can also change the value of a keyword for the duration of a request by including the keyword and its temporary value in the request (with a command other than **set**); the keyword must be one of those associated with the command.

Forgetting that you have changed the value of a keyword in your active keyset can cause problems. If you begin receiving error responses you do not anticipate, check the contents of your working directories and any working directory passwords you have set to be sure they are what you currently need.

For more information about keywords see Chapter 2, Section 2.2.3. Table 2.4 provides a summary of keyword functions and default values.

A goal of this manual is to to provide the functionality required by most users in the shortest and simplest way possible. To meet this goal, the use of keywords in requests is limited in two ways.

- First, this manual does not cover all keywords; the default values for those not covered should meet the needs of most users.
- Second, for each command, not all associated keywords are discussed. For example, the keyword `nrel=` can be used with several commands including `copy` (see Table 7.2). In this manual, however, it is used only with `modify`. Thus, to copy `file1` to `file2` and give `file2` a release date seven days later than the one in effect for `file1`, you are instructed to enter

  ```
  copy file1 to file2
  modify file2 nrel=007
  ```

For a list of all the keywords that are associated with each command see Table 7.2; Table 2.5 shows the subset of those options that is covered in this manual. If you find you need more detailed information about the use of keywords than is presented in this manual, see the *Common File System CFS Interface Reference*, which is available for reference in the Programming Assistance offices.

The keyword descriptions are presented in alphabetical order and follow the format outlined below.

- **The keyword.**
- **A functional description.**
- **Shortest keyword abbreviation:** The shortest acceptable keyword abbreviation, if one is allowed.
- **Commands used with:** The commands with which the keyword can be specified in a request (actually, a subset of the full list as explained above).

- **Keyword value(s):** An explanation of the non-null values that may be associated with the keyword. A null value is an option whenever the default value of the keyword in the active keyset is a null.
- **Default value:** The default value in the active keyset when you invoke **cfsi**.
- **Additional comments:** Additional comments, if any. If multiple values can be specified with the keyword (see Chapter 2, Section 2.2.3), this fact is noted.
- **Examples:** Examples demonstrating use of the keyword. In all cases, two conditions are assumed unless otherwise stated:

  — **dir0**, the default working directory, contains your userid root (i.e., */userid*) and

  — the nodes involved are not password-protected.

  In addition, if a request copies a file to or from a UNICOS directory, it is assumed that you invoked **cfsi** from that directory.

  The request shown in an example may not be the only one that can be used to accomplish the described function.

  As explained in Chapter 2, a backslash (\) at the end of an input line indicates that the request is continued on the next input line.

When entering passwords under **cfsi**, be sure to protect your screen from observation by others. If you are requesting information that includes passwords and have no need to view the passwords, be sure that the keyword **lpw=** is set to **off** (the default).* If you display the passwords (**lpw=on**), protect your screen from observation by others.

* **lpw=off** does not currently mask passwords when keyword values are listed with **set** and **show**. This problem should be corrected in a future version of CFS.

## 11.1 `aval=`

Adds a new validation entry to the *user validations* field of an existing file descriptor or directory node. Makes an initial change in the *master user validation* entry for a non-root node. (To change any non-owner user validation entry or to change the master user validation entry for a root or another node for which a change has already been made, use `cval=`; to delete an entry, use `dval=`.)

**Shortest keyword abbreviation:** `av=`

**Commands used with:** `modify`

**Keyword value(s):** *userid/rights/pw/modifier*

where

| | |
|---|---|
| *userid* | is the userid being assigned rights. |
| *rights* | are any combination of the characters that specify the following access rights: |

| | | | | | |
|---|---|---|---|---|---|
| `r` | (read) | `i` | (insert) | `a` | (append) |
| `w` | (write) | `b` | (bestow) | `e` | (execute) |
| `m` | (modify) | | | | |

If you enter a null in this field (i.e., no character or space between the first and second slashes), the specified user will have no access rights. For a discussion of access rights, see Chapter 8, Section 8.1.3.

*pw* is a password of five to eight alphanumeric characters to be required of the specified user for access. If no password is required, use a minus sign (-) in this field. Use a null field to show that the password from a higher level in the tree is to be used. A null password cannot be specified for a root, because a root is the highest level in a tree.

Precede each password you enter in a request with a caret (^). See Chapter 8, Section 8.1.4 for more information about passwords.

Be sure to protect your screen from view by others when entering passwords in a request.

*modifier* is one of the following characters to specify the access rights accumulation modifier (see Chapter 8, Section 8.1.2):

| | | | | | |
|---|---|---|---|---|---|
| `s` | (set) | `o` | (or) | `a` | (and) |

**Default value:** null

**Additional comments:**

You can specify multiple values with `aval=`.

CFS creates a master user validation entry in the *master user validation* field for a root node as described in Chapter 8, Section 8.1.1. That entry is inherited by all descendant nodes and appears in the *master user validation* field for those nodes. To change the entry for a descendant node, you do not initially use `cval=` as might be expected. Instead, you use `aval=` set to the desired new value. `aval=` causes the new entry to appear in the *master user validation* field for the node and also creates a duplicate entry in the *user validations* field. Once this duplicate entry has been created, you use `cval=` to make any future changes.

You cannot use `aval=` to remove an owner's modify access to any node.

Because write and/or modify access allow a user so much control, it is good practice not to grant these rights to other users for your root nodes. Grant these rights to other nodes only when special circumstances require.

**Examples:**

- Permit userids 010203 and 040506 to have read access to your root directory without a password.

```
modify aval=(010203/r/-/s 040506/r/-/s) \
/userid
```

- Assume that the password `lookat` is required for access to your userid root and, therefore, to the subdirectory defined by the path */userid/*`uses`. Retain the password `lookat` for access to */userid*, but require the password `reopen` for your access to `uses`.

```
modify /userid/uses pw=^lookat \
aval=userid/rewaibm/^reopen/s
```

The *master user validation* field of your userid root still contains the entry *userid*/**rewaibm/lookat/s**. The *master user validation* and *user validations* fields for the node **uses** each contain the entry *userid*/**rewaibm/reopen/s**. **cval=** must be used for future changes to the master user validation entry for the node **uses**.

## 11.2 `charge=`

Sets the C&TD computer charge number for a root node when the node is created (i.e., sets a value for the node's *charge* field). (To change the charge number for a root node, use `ncharge=` with the `modify` command.)

**Shortest keyword abbreviation:** `ch=`

**Commands used with:** `create`

**Keyword value(s):**

A valid, five-digit C&TD computer charge number preceded by three zeros.

**Default value:**

The default charge number for your UNICOS session.

**Additional comments:**

All charges associated with files stored under the node are made to the charge number specified for the node.

You must create a separate root for each charge number you want to use.

**Examples:**

- Create a root directory named `tests` with charge number 12345 that is different from the default.

```
create tests charge=00012345
```

## 11.3 cval=

Changes the rights, password, and/or modifier in a user validation entry listed in the *user validations* field for a file descriptor or directory node. Also makes changes to the *master user validation* field for a root node and for a node for which an initial change has been made in that field. (To add a non-owner validation entry or make an initial change in a non-root master user entry, use `aval=`; to delete an entry, use `dval=`.)

**Shortest keyword abbreviation:** `cv=`

**Commands used with:** `modify`

**Keyword value(s):** *userid/rights/pw/modifier*

where

*userid* is the userid being assigned rights.

*rights* are any combination of the characters that specify the following access rights:

| | | | | | |
|---|---|---|---|---|---|
| `r` | (read) | `i` | (insert) | `a` | (append) |
| `w` | (write) | `b` | (bestow) | `e` | (execute) |
| `m` | (modify) | | | | |

If you enter a null in this field (i.e., no character or space between the first and second slashes), the specified user will have no access rights. For a discussion of access rights, see Chapter 8, Section 8.1.3.

*pw* is a password of five to eight alphanumeric characters to be required of the specified user for access. If no password is required, use a minus sign (-) in this field. Use a null field to show that the password from a higher level in the tree is to be used. A null password cannot be specified for a root, because a root is the highest level in a tree.

Precede each password you enter in a request with a caret (^). See Chapter 8, Section 8.1.4 for more information about passwords.

Be sure to protect your screen from view by others when entering passwords in a request.

*modifier* is one of the following characters to specify the access rights accumulation modifier (see Chapter 8, Section 8.1.2):

| | | | | | |
|---|---|---|---|---|---|
| `s` | (set) | `o` | (or) | `a` | (and) |

**Default value:** null

**Additional comments:**

You can specify multiple values with `cval=`.

CFS will not permit you to remove modify access from a master user validation entry.

You can use `cval=` to perform two basic functions:

- changing the master user validation entry for a root node, and
- changing an entry that appears in the node's *user validations* field. If the entry is for the owner's userid, the corresponding entry in the *master user validation* field is also changed.

If a validation has been inherited from a related, higher-level node and therefore does not appear as an entry in the *user validations* field for the node, use `aval=` to change it for the current node (see last example in Section 11.1).

Because write and/or modify access allow a user so much control, it is good practice not to grant these rights to other users for your root nodes. Grant these rights to other nodes only when special circumstances require.

**Examples:**

- Change the password required for you to access your root node **trials** (from **oldpwd** to **newpwd**).

```
modify /trials pw=^oldpwd \
cval=userid/rewaibm/^newpwd/s
```

- Add insert access to the read access already granted to userid 010203 for the root node **examples**.

```
modify /examples cval=010203/ri/-/s
```

## 11.4 `dir`*n*=

Can provide part or all of a path for a CFS request.

**Shortest keyword abbreviation:** *dn*=

**Commands used with:**

**`add, copy, delete, get, list, modify, move, remove, replace, save, set, store`**

**Keyword value(s):** a path

**Default value:**

| | |
|---|---|
| `dir0` | */userid* (your userid root node) |
| `dir1 - dir9` | null |

**Additional comments:**

Each `dir`*n* represents a working directory, where *n* is one of the digits 0 through 9. Use the `dir`*n*= keywords to define paths for working directories. When a working directory is referenced in a request, its contents are used as directed by CFS in forming the complete path required for the request.

Use the *n'* notation to refer to a working directory where *n* is the *n* in `dir`*n*. If a password is required to access the path stored in `dir`*n*, the password can be stored as a working directory password (pw*n*) using the keyword pw*n*=. Once stored, the password will be used automatically by CFS whenever the corresponding `dir`*n* is referenced in a request.

`dir0` is the default working directory and can be abbreviated as `dir` or `d` in addition to `d0`. When no *n'* reference is made in a request where one is required, CFS uses the contents of `dir0`.

This manual deals only with working directories containing paths that begin with a root (i.e., are preceded by a slash). Sometimes a working directory is used to supply the initial portion of a path in a request and sometimes to supply the complete path.

The path contained in a working directory cannot be of the form *workerfile:path*.

For a discussion of paths, see Chapter 3. Chapter 6 deals with working directories and working directory passwords.

Examples:

- Save the file **vegetables** under your userid root node using the default path stored in **dir0**.

```
save vegetables
```

- Save the files **first** and **second** under the subdirectory defined by the path */userid/*`list` using **dir7**.

```
set dir7=/userid/list
save 7'first
save 7'second
```

  The **set** request established a value for **dir7=** in your active keyset that can be referred to in later requests.

- In the above example, assume that the password **letmein** is required to access **list**.

```
set dir7=/userid/list pw7=^letmein
save 7'first
save 7'second
```

- Copy the file **exams** defined by the path */userid/*`misc` to the subdirectory **school** defined by the path **/outside/school**. The password **letmein** is required for you to access **misc** and **congrats** to access **school**.

```
copy 3'exams to 4'exams pw3=^letmein \
dir3=/userid/misc pw4=^congrats \
dir4=/outside/school
```

  The values for **dir3=**, **dir4=**, **pw3=**, and **pw4=** are set only for the duration of the request.

## 11.5 dval=

Deletes a validation entry for the specified userid from the *user validations* field of a file descriptor or directory node. (To add a validation entry or to make an initial change in a master user validation for a non-root node, use `aval=`; to make changes to the *master user validation* field for a root node or for the *user validations* field for any node, use `cval=`.)

**Shortest keyword abbreviation: dv=**

**Commands used with:** `modify`

**Keyword value(s):**

The userid of the user validation entry being deleted.

**Default value: null**

**Additional comments:**

You can specify multiple keyword values with `dval=`.

If you set `dval=` to *userid/rights/pw/modifier*, CFS uses the userid and ignores the remainder of the input.

You cannot delete the master user validation entry from a root node.

When you use `dval=` to delete the master user validation entry from a non-root node, the current entry in the *master user validation* field for the node is replaced by a copy of the entry for the parent node. In addition, if there is a duplicate entry for the owner's userid in the *user validations* field for the node, that entry is deleted.

See Chapter 8 for more information about validation entries.

**Examples:**

- Delete the validation entry for userid 010203 from your userid root.

```
modify /userid dval=010203
```

or

```
modify - dval=010203
```

- Delete the validation entries for userids 010203 and 040506 from the node defined by the path **/a/b/c**.

```
modify /a/b/c dval=(010203 040506)
```

- Perform the above procedure, assuming that the root node **a** is protected by the password **whoisit**.

```
modify /a/b/c dval=(010203 040506) \
pw=^whoisit
```

## 11.6 grp=

Specifies the choice of physical storage group at the time a file is copied from UNICOS to CFS; either locks a file into group A or B (i.e., sets the value of the node's *group* field to **a** or **b**) or allows CFS to select the storage group. (To lock a file into the current storage group when no storage group has been previously set, use **setgrp=** with the **modify** command; to set the storage group for a file being created using the **copy** command, use **ngrp=**.)

**Shortest keyword abbreviation: gr=**

**Commands used with: save, store**

**Keyword value(s):**

One of three characters as follows:

| | |
|---|---|
| a | to specify storage group A |
| b | to specify storage group B |
| - | to set the keyword value to null (i.e., to specify that CFS will select the storage group) |

**Default value:** null (CFS selects the group)

**Additional comments:**

Because the storage devices used by CFS are extremely reliable, there are no automatic system backups of files stored on CFS. However, the storage devices are divided into two physically separate groups, so you can back up mission-essential files by storing duplicate copies, one on each group. If a file has been stored (using **save** or **store**) with **grp=** set to **a** or **b**, it will not change groups during a CFS-initiated migration or execution of a **replace** request. Note that backup copies stored under the same directory node must have different file names (e.g., **test** and **test.bak**).

Unless you are backing up mission-essential files, you should use the default value for **grp=** (i.e., do not use **grp=** in the request) so that CFS can store and migrate files to balance the load between storage devices.

**Examples:**

- Save the file **test** in the root directory **compile** and specify the storage group A.

```
save /compile/test grp=a
```

- **`test`**, which was just saved on storage group A, is a mission-essential file. Back it up by saving another copy of **`test`** in the same directory but on storage group B. Call this copy **`test.bak`**.

```
save test:/compile/test.bak grp=b
```

## 11.7 `info=`

Stores text information in the *info* field of a root or subdirectory node to help identify the contents of the associated tree. (Change the contents of the *info* field for a node or add identifying text information to a file node using `ninfo=` with the `modify` command.)

**Shortest keyword abbreviation:** `inf=`

**Commands used with:** `add, create`

**Keyword value(s):**

One to 80 characters of text information enclosed in double quotes (" ").

**Default value:** null

**Additional comments:**

You can list the *info* field of a node by using the `lo=i` or `lo=g` option with the `list` command. You may find that an 80-character field is not displayed properly on your terminal. If so, try limiting the field to 71 or 72 characters.

CFS will not accept uppercase characters in the text for `info=`.

Including descriptive information in a node can help identify files on CFS, saving the trouble of transferring the files to UNICOS for examination.

**Examples:**

- Create a root directory `reports` and include text describing the purpose to which the node will be put.

  ```
  create /reports info="final drafts \
  of 1988 monthly and quarterly reports".
  ```

- Add the subdirectory `monthly` under the node just created and include an appropriate text field.

  ```
  add /reports/monthly info="final drafts \
  of 1988 monthly reports"
  ```

## 11.8 kpw=

Specifies the password that will be required for your access to a saved keyset when used with the `keep` command. Supplies the password required for your access to a saved keyset when used with the `free`, `show`, or `adopt` command.

**Commands used with: `adopt, free, keep, show`**

**Keyword value(s):**

A password of five to eight alphanumeric characters.

**Default value:** null

**Additional comments:**

Because you are the only user who can access your saved keysets, you may not want to bother with using keyset passwords.

When entering a keyset password in a request, you must precede the password with a caret (^).

Protect your screen from view by others when passwords are displayed.

For information about saved keysets, see Chapter 7 and Chapter 8, Section 8.3.

**Examples:**

- Save your active keyset as **key2** and specify that the password **sesame** will be required for access to the keyset.

```
keep key2 kpw=^sesame
```

- Later, make **key2** your active keyset.

```
adopt key2 kpw=^sesame
```

- Display the contents, including passwords, of the keyset **key2** saved above.

```
show key2 so=o kpw=^sesame lpw=on
```

## 11.9 `lo=`

The list option (`lo=`) keyword selects one or more options for the **`list`** command. Each option displays selected information fields (attributes) for the specified node.

**Commands used with: `list`**

**Keyword value(s):**

One or more of the following characters representing the indicated information:

| | |
|---|---|
| **a** | accounting information; valid only for root nodes |
| **d** | immediate descendants; valid only for directory nodes |
| **g** | general information |
| **u** | user validation entries |
| `i` | contents of *info* field |
| **s** | size of file in bits; valid only for file descriptor nodes |

**Default value:**

| | |
|---|---|
| **d** | for a directory node |
| **g** | for a file descriptor node |

**Additional comments:**

There is no list option that will display the names of your root nodes. However, you can list your root names by following the procedure described in Chapter 9, Section 9.4.

The values in some node fields (e.g., *date last modified*) are supplied by CFS. Others (e.g., *node name, info*) depend on the use of keywords and parameters in the appropriate requests. In either case, fields to which no specific values have been assigned contain null values (i.e., are blank).

The options `i` and **s** produce components of the information provided by the **g** option.

You may specify multiple values with the `lo=` keyword using the syntax described on the next page.

Read access to a node is required to execute most list options for the node; Modify access is required for list option **u**.

Appendix A contains some additional information about the attributes associated with root, subdirectory, and file nodes.

**Examples:**

The examples in the bullet list that follows show samples of the type of output that is produced when various list options are specified with various parameter types. The general form of the request is

`list` *parameter* `lo=`*option*

as in

`list` */userid/*`test lo=g`

or, for multiple list options,

`list` *parameter* `lo=`*option1option2...*

as in

`list` */userid/*`test lo=gi`

Note that the syntax for specifying multiple list options differs from that used to specify multiple values for other keywords (parentheses and spaces are omitted).

- `lo=g`
  - For a root node.

```
node name: reports
node type: root directory
last modified: 88/09/20 14:16
last user to modify: 010203
created: 88/09/10 12:10
user created by: userid
info: drafts of 1988 reports
partition: green
cl: u
oid:
```

  - For a subdirectory node.

```
node name: monthly
node type: subdirectory
last modified: 88/09/20 14:16
last user to modify: 010203
created: 88/09/10 12:10
user created by: userid
info: drafts of monthly reports
partition: green
cl: u
```

— For a file descriptor node.

```
node name: jan
node type: file descriptor
last modified: 88/09/20 14:16
last user to modify: userid
created: 88/09/10 12:10
user created by: userid
info:
partition: green
cl: u
original system: unix
storage type: tape offline
group: a
use: w
size:     884736
release date: never
last written: 88/09/20 14:16
last read:
number of accesses:     1
compression: off
```

- `lo=d`

  — For a directory (here, a root node named **`reports`**). Note that in the response, subdirectories are labeled with the identifier *dir* to distinguish them from files.

```
node name: reports
node type: root directory
descendants:
suba      dir
subb      dir
filea
fileb
```

- `lo=u` (Modify access to the specified node required)
  - For a root node.

```
node name: reports
node type: root directory
master user validation:
userid/rewaibm/-/s
user validations:
010203/r/-/s
040506/r/-/s
```

  - For a non-root node (here, a subdirectory named **`testruns`**). Note that only *user validations* that have been written for the specified node are listed; the user validations for related nodes at higher levels of the tree may affect access to the node. The master user validation was inherited from the parent node because no duplicate entry appears in the *user validations* field.

```
node name: testruns
node type: subdirectory
master user validation:
userid/rewaibm/-/s
user validations:
040506/r/-/s
070809/r/-/s
```

- `lo=i`
  - For a file descriptor node (here, named `comments`).

```
node name: comments
node type: file descriptor
info: energy project review comments
```

- `lo=a` (see Chapter 5 for information about CFS charges)
  - For a root node.

```
node name: project
node type: root directory
oid:
charge: 00012345
space time product starts: 86/10/07 12:34
online space: 134.959 Mb
online files: 43
online space time product: 1572.8 Mb days
offline space: 4.615 Mb
offline files: 80
offline space time product: 928.6 Mb days
```

## 11.10 `lpw=`

Allows or disallows the display of passwords in responses containing validation entries, working directory passwords, and keyset passwords.

**Commands used with: `list`, `set`*, `show`***

**Keyword value(s):**

One of the following words to select the indicated option:

| | |
|---|---|
| `on` | list passwords |
| `off` | do not list passwords |

**Default value: `off`**

**Additional comments:**

If `lpw=off`, a string of eight percent signs (i.e., `%%%%%%%%`) indicates the presence of a password.

Set `lpw=` to `on` only when it is necessary to view a password. Protect your screen any time a password is displayed. If you use the `set` command to change the value of `lpw=` to `on` in your active keyset, remember to change the value back to `off` as soon as possible.

**Examples:**

- Display the validation entries for the subdirectory **`testruns`**, which requires the password **`sesame`** for access, first displaying the password, then supressing the password. The responses are shown for illustration.

* `lpw=off` does not currently work when used with `set` or `show` to display keyword values. However, this problem should be corrected in the near future.

```
list /userid/testruns pw=^sesame lpw=on lo=u
```

```
node name: testruns
node type: subdirectory
master user validation:
userid/rewaibm/sesame/s
user validations:
040506/r/sesame/s
010203/r/sesame/s
```

```
list /userid/testruns pw=^sesame lo=u
```

```
node name: testruns
node type: subdirectory
master user validation:
userid/rewaibm/%%%%%%%%/s
user validations:
040506/r/%%%%%%%%/s
010203/r/%%%%%%%%/s
```

- Check to see what password is associated with `dir3`.

```
set pw3 lpw=on
```

```
pw3     openup
```

Note that using **`lpw=on`** with the **`set`** command changes the value of `lpw` in the active keyset to **`on`**. To change it to **`off`** to prevent passwords from being unintentionally displayed, enter

```
set lpw=off
```

## 11.11 `ncharge=`

Establishes a new C&TD computer charge number for a root node (i.e., sets a new value for the node's *charge* field).

**Shortest keyword abbreviation: `nch=`**

**Commands used with: `modify`**

**Keyword value(s):**

A valid, five-digit C&TD computer charge number preceded by three zeros.

**Default value:** null

**Additional comments:**

When you change the charge number associated with a root, all subsequently incurred charges associated with files stored under the root are made to the new number.

Use the **`list`** command with the **`lo=a`** option to check the current charge number for a root node.

See Chapter 5 for information about CFS charges.

**Examples:**

- Change the charge number associated with your userid root to 34567.

```
modify /userid ncharge=00034567
```

- Assign the charge number 67899 to the root named **reports**, which requires the password **sesame** for access.

```
modify /reports ncharge=00067899 \
pw=^sesame
```

## 11.12 **ngrp=**

Sets the storage group for the new file in a `copy` request (i.e., establishes a value for the *group* field for a file descriptor node). (To lock a file into the current storage group when no group has been set previously, use **setgrp=** with the `modify` command; to set a storage group for a file being created using **save** or **store**, use **grp=**.)

**Shortest keyword abbreviation: ngr=**

**Commands used with:** `copy`

**Keyword value(s):**

One of four values as follows:

| | |
|---|---|
| a | to specify storage group to A |
| b | to specify storage group to B |
| n | to allow CFS to select the storage group; the *group* field for the node will contain a null* |
| o | to assign the new file to the group opposite to that of the source file; also locks the source file in to the group on which it resides if the group has not been set previously |

**Default value:** null (CFS selects the group)

**Additional comments:**

If **ngrp=** is not used with the `copy` command, the value in the *group* field of the source file node is assigned to the new file node.

Because the storage devices used by CFS are extremely reliable, there are no automatic system backups of files stored on CFS. However, the storage devices are divided into two physically separate groups, so you can back up mission-essential files by storing duplicate copies, one on each group. If you specify a storage group (A or B) for a file, it will not change groups during subsequent CFS-initiated migrations or executions of **replace** requests.

* This value does not work; the value in the group field for the source node is copied to the new node.

Note that backup copies stored under the same directory node must have different file names (e.g., **test** and **test.bak**). Use the **list** command with the **lo=g** option to determine what group, if any, has been set for a file.

**Examples:**

- The file **results**, a direct descendant of your userid root, needs to be backed up. The backup copy, which will also be stored directly under your userid root, will be called **results.bak**.

  ```
  copy results to results.bak ngrp=o
  ```

  To check the group assignments, enter the following two requests and check the contents of the *group* field in each case.

  ```
  list results lo=g
  list results.bak lo=g
  ```

  As an alternative to the **copy** request above, you can first lock **results** into the storage group on which it is currently stored

  ```
  modify results setgrp=on
  ```

  then enter

  ```
  list results lo=g
  ```

  to determine what that group is. Finally, assuming **results** is now assigned to storage group A, execute a copy assigning **results.bak** to storage group B.

  ```
  copy results to results.bak ngrp=b
  ```

## 11.13 `ninfo=`

Replaces the contents (text or null) of the *info* field of a file descriptor or directory node with new text. (Use the **`info=`** keyword with **`create`** or **`add`** to place text in the *info* field when creating a root or subdirectory node.)

**Shortest keyword abbreviation:** `ninf=`

**Commands used with:** `modify`

**Keyword value(s):**

One to 80 characters of text information enclosed in double quotes (" ").

**Default value:** null

**Additional comments:**

Text is usually placed in the *info* field of a file descriptor node to describe the contents of the associated file and in a directory node to describe the category of files stored in the tree or subtree.

Including descriptive information in a node can help identify files on CFS, saving the trouble of transferring the files to UNICOS for examination.

CFS will not accept uppercase characters in the text for **`ninfo=`**.

Use the **`lo=i`** or **`lo=g`** option with the **`list`** command to display the *info* field of a node. You may find that an 80-character field is not displayed properly on your terminal. If so, try limiting the field to 71 or 72 characters.

**Examples:**

- Change the *info* field for the root **`reports`**.

```
modify /reports ninfo="monthly reports \
for 1989"
```

- Place information in the *info* field for the file descriptor node associated with the file **`comments`**, which is defined by the path */userid/*`comments`.

```
modify /userid/comments ninfo="comments \
from review of progress report"
```

## 11.14 `nname=`

Changes the name of a named root, subdirectory, or file descriptor node. The new name replaces the former name in the *node name* field of the node.

**Shortest keyword abbreviation:** `nn=`

**Commands used with:** `modify`

**Keyword value(s):**

A name you select; may be up to 16 characters in length and incorporate any of the following:

`a-z`, `A-Z`, `0-9`, `$`, `%`, `*`, `+`, `-`, `.`, and `_`.

The name must not be a path and must not be preceded by a slash (/). The new name for a root cannot be one already in use on CFS.

**Default value:** null

**Additional comments:**

Renaming a node does not change its owner. If you need to assume ownership of someone's CFS files (e.g., you are taking over the job of someone who is leaving the company), Programming Assistance can provide assistance.

You cannot rename someone else's root to your userid.

You should not change the name of a root belonging to another user.

**Examples:**

- Change the name of your root node **reports88** to **reports89**.

  ```
  modify /reports88 nname=reports89
  ```

- Change the name of the file (and its associated file descriptor node) **testorig**, defined by the path */userid*/project/testorig, to **testnew**.

  ```
  modify /userid/project/testorig \
  nname=testnew
  ```

## 11.15 `nrel=`

Defines a new release date for a file (i.e., places a new value in the *release date* field for a file descriptor node).

**Commands used with:** **`modify`**

**Keyword value(s):**

The date you want the file to be released for purging (deletion) by CFS. Two date formats are allowed:

| | |
|---|---|
| *yy/mm/dd* | where *yy* is the year, *mm* the month, and *dd* the day; and |
| *nnn* | where *nnn* is a number of days (001 through 365) to be added to the current release date to create a new release date. |

The file is released at midnight on the date you specify.

You can set a file to be released on any date up to 99/12/31. A date of 99/12/31 is considered by CFS to be *never* (i.e., the file will not be released).

**Default value:** null

**Additional comments:**

When you create a file, CFS assigns it an initial release date of *never* (i.e., 99/12/31).

Use the **`list`** command with the **`lo=g`** option to determine the current release date for a file.

If you copy a file from one location on CFS to another using the `copy` command, the new copy will have the release date of the source file.

**Examples:**

- Tell CFS to release the file named **info.temp** one week from the current day (August 3, 1988). **info.temp** is defined by the path */userid/***info.temp**.

```
modify /userid/info.temp nrel=88/08/10
```

  or

```
modify /userid/info.temp nrel=007
```

- Change the release date for the file defined by the path **/proposal/quotes** from the default to December 6, 1989.

```
modify /proposal/quotes nrel=89/12/06
```

## 11.16 `nuse=`

Defines a revised estimate of file activity (changes the value of the *use* field in a file descriptor node).

**Shortest keyword abbreviation:** `nus=`

**Commands used with:** `modify`

**Keyword value(s):**

One of the following characters representing an estimate of the frequency with which you intend to access the specified file:

| | |
|---|---|
| `d` | daily |
| `w` | weekly |
| `m` | monthly |
| `x` | file expected to be active for a few days, then may be stored on tape by the CFS migration program |
| `a` | file expected to be accessed infrequently; will be stored on disk initially and transferred to tape overnight |

**Default value:** null

**Additional comments:**

When you create a file, CFS assigns it an initial estimate of activity of `w`.

CFS decides whether to place files online on disk or offline on tape. All files are first placed on disk and are only moved to tape if not accessed within a system-set time period. The `nuse=` keyword allows you to influence the placement of your files. If you will use a file frequently, it will automatically be retained on disk and you do not have to be concerned about the setting of this attribute in the file node. However, if you know you will not be using a file very often, you can have it moved to tape overnight by using the `a` option.

When you copy a file from one location to another on CFS, the estimated file activity of the original file node is copied to the new file node.

To determine whether a file is stored on tape or disk, enter

```
list *path* lo=g
```

where *path* is the complete path for the file. Check the *storage type* field of the response.

**Examples:**

- Force the file **data.bak**, which you intend to retain only as an emergency backup, to be stored on tape. **data.bak** is a direct descendant of your userid root node.

  ```
  modify data.bak nuse=a
  ```

## 11.17 **pw*n*=**

In a request with most commands, can supply the password that allows access to the path specified as the parameter; with the `create` command, specifies the password that will be required of the owner to access the new root.

**Commands used with:**

**`add, copy, create, delete, get, list, modify, move, remove, replace, save, set, store`**

**Keyword value(s):**

A password of five to eight alphanumeric characters.

**Default value:** null

**Additional comments:**

**pw*n***, where *n* is one of the digits 0 through 9, is known as a working directory password (see Chapter 6, Section 6.4). The password is specified using the **pw*n*=** keyword. `dir`*n* represents a working directory. Working directories are described in Chapter 6. When `dir`*n* is used explicitly or by default in a request, the password set for the correspondingly numbered **pw*n*=** keyword is used automatically.

You can abbreviate **pw0=** as **pw=**.

When entering a password in a request, precede the password with a caret (^).

To specify the password when a path is specified in full in a request, use **pw=**^*password.*

Be sure to protect your screen from view by others when entering passwords.

When two paths are required in the parameter for a command (i.e., *path1* `to` *path2* for `move` and `copy`), two **pw*n*=** keywords may be required:

— one to allow access to the source file and
— one to allow access to the new parent node.

**Examples:**

- Retrieve a copy of the file described by the path */userid/*`reports/jan`. The password `mypass` is required to access `reports` and its descendants.

```
get /userid/reports/jan pw=^mypass
```

  or

```
set dir0=/userid/reports pw0=^mypass
get jan
```

- Place a copy of the file `newdata`, a direct descendant of your userid root, in the subdirectory defined by the path `/project/firstrun`. Call the new copy `newdata.cop`. Your userid root is protected by the password `mypass` and `firstrun` by the password `sesame`.

```
copy newdata to 2'newdata.cop \
dir2=/project/firstrun pw0=^mypass \
pw2=^sesame
```

  or

```
set pw0=^mypass
copy newdata to 2'newdata.cop \
dir2=/project/firstrun pw2=^sesame
```

  or

```
set pw0=^mypass pw2=^sesame \
dir2=/project/firstrun
copy newdata to 2'newdata.cop
```

- Change the password you use to access the root node `roota` from `oldpass` to `newpass`.

```
modify /roota pw=^oldpass \
cval=userid/rewaibm/^newpass/s
```

- Create your userid root node and specify that you will have to supply the password **sesame** to access the node.

  ```
  create pw=^sesame
  ```

  If you subsequently enter

  ```
  list lo=u pw=^sesame lpw=on
  ```

  you will find that the password **sesame** has been incorporated in the *master user validation* field of the root node.

  ```
  node name: userid
  node type: root directory
  master user validation:
  userid/rewaibm/sesame/s
  user validations:
  ```

## 11.18 `setgrp=`

Locks a file into the current storage group (i.e., sets a value for the node's *group* field) if a storage group was not specified when the file was stored. (To set the storage group for a file being created using the `copy` command, use `ngrp=`; to set a storage group for a file being created using `save` or `store`, use `grp=`.)

**Shortest keyword abbreviation:** `setg=`

**Commands used with:** `modify`

**Keyword value(s):**

| | |
|---|---|
| `on` | locks the file into the storage group on which it currently resides* |

**Default value:** `off`

**Additional comments:**

Because the storage devices used by CFS are extremely reliable, there are no automatic system backups of files stored on CFS. However, the storage devices are divided into two physically separate groups, so you can back up mission-essential files by storing duplicate copies, one on each group. If you specify a storage group (A or B) for a file, it will not change groups during subsequent CFS-initiated migrations or executions of `replace` requests. Note that backup copies stored under the same directory node must have different file names (e.g., `test` and `test.bak`).

Use the `list` command with the `lo=g` option to determine what group, if any, has been set for a file.

* There is currently no option for "unlocking" a file once a group has been set.

**Examples:**

- You have stored the file `info` under your userid root without specifying a storage group. You have determined that the file contains mission-essential information and, thus, must be backed up by ensuring that a copy exists on each storage group. First, lock in `info` on its current storage group as follows

  ```
  modify info setgrp=on
  ```

  Next, find out what that storage group is by entering

  ```
  list info lo=g
  ```

  and checking the value in the *group* field. Assume the group is A. Now create a copy of `info`, calling it `info.bak` and placing it on storage group B.

  ```
  copy info to info.bak ngrp=b
  ```

## 11.19 so=

Determines what saved keysets or keyset information will be displayed in response to a `show` command.

**Commands used with:** `show`

**Keyword value(s):**

One of three single-character options as follows:*

| | |
|---|---|
| `o` | outputs for the specified keyset, its name, creation date, date last saved, and its full set of keywords and their values |
| `n` | outputs for each of your saved keysets the keyset name, creation date, and date last saved |
| `a` | outputs for each of your saved keysets, the keyset name, creation date, date last saved, and its full set of keywords and their values |

**Default value:** `n`

**Additional comments:**

Passwords included in keysets are displayed only if the keyword `lpw=` is set to `on`.**

**Examples:**

- Display a complete set of information about your userid keyset.

```
show userid so=o
```

or

```
show so=o
```

* Currently, only the `o` option works properly. For consistent results, include `so=o` and a parameter in each `show` request.

** `lpw=off` does not currently work with the `show` command. Therefore, passwords are always displayed. This condition should be corrected in the future.

- Display complete information about the saved keyset **newkey**, which is protected by the keyset password **access**.

```
show newkey so=a kpw=^access
```

# Appendix A
# CFS Node Contents

Each type of node in a CFS tree structure contains specific information fields as described in Chapter 3, Section 3.1. This appendix provides an overview of the information contained in each type of node (i.e., root, subdirectory, and file descriptor). For information about viewing node fields using the **`list`** command with the `lo=` keyword, see Chapter 9, Section 9.10 and Chapter 11, Section 11.9.

**Root Node Information**

- The node name and type.
- The master user (i.e., owner's) validation entry, which consists of the following:
  - — The userid of the user who is the owner of the tree. The owner is usually the person who created the root.
  - — All the owner's access rights. The owner of the tree initially has all access rights to the root and all nodes that are added to the tree.
  - — A password, if one has been specified for the owner's access.
  - — An access rights modifier; **`s`** is the default.
- Access control information (i.e., user validation entries) about any other userids for which access rights have been specified (see the keywords **`aval=`**, **`cval=`**, and **`dval=`** in Chapter 11). The access granted may also affect access to nodes in lower levels of the tree.
- A five-digit CT&D computer charge number. All charges associated with all files in the tree are billed to this number.
- Security and administrative information, including the date and time the root was created and last modified, the userid(s) of the user(s) who created and last accessed the root, the node's security partition (always green on the K-25 CFS) and classification level (always unclassified on the K-25 CFS), and accounting information (i.e., online and offline space use, number of files, cumulative space-time product).
- A descriptive text field, if one has been entered using the **`info=`** or **`ninfo=`** keyword. The text usually explains the use to which the node is being put.
- Owner identification information entered by the user using the **`oid=`**, or **`noid=`** keyword. (Note that these keywords are not described in this manual; the field will contain a null unless you entered information in the field using **`mass`** on CTSS.)
- The names of all the root's immediate descendants. Subdirectory nodes are labeled *dir*.

**Subdirectory Node Information**

- The name and type of the node.
- A descriptive text field, if one has been entered using the `info=` or `ninfo=` keyword. The text usually explains the use to which the node is being put.
- The names of all the nodes's immediate descendants. Subdirectory nodes are labeled *dir.*
- The master user (i.e., owner's) validation entry, which consists of the following:
  - — The userid of the user who is the owner of the node.
  - — All the owner's access rights.
  - — A password, if one has been specified for the owner's access.
  - — An access rights modifier; `s` is the default.
- Access control information (i.e., user validation entries) about any other userids for which access rights to the node have been specified (see the keywords `aval=`, `cval=`, and `dval=` in Chapter 11). To determine the protection for a node, the user may have to examine and, if the access rights modifiers `o` or `a` have been used, accumulate the protection for the root, any intermediate nodes, and the node itself. The access granted to a subdirectory node may affect access to nodes at lower levels of the subtree.
- Security and administrative information, including the date and time the node was created and last modified, the userid(s) of the user(s) who created and last accessed the node, and the node's security partition (always green on the K-25 CFS) and classification level (always unclassified on the K-25 CFS).

**File Descriptor Node Information**

Note that a file descriptor node does not contain the file itself, but does contain information about the file and a pointer to the actual file storage location. The distinction between a file descriptor node and a file is usually not important to users. Both the node and the file have the same name. When a user specifies the name in a request, CFS is able to determine whether it is the file or node that is being referenced. For example,

`get` *filename*

results in retrieval of a copy of the specified file while

`list` *filename* `lo=s`

results in retrieval of file-size information from the corresponding file node.

The information included in a file descriptor node includes

- The node name and node type.
- A descriptive text field, if one has been entered using the **ninfo=** keyword. The text is usually used to provide information about the content of the associated file.
- The location of the file in CFS storage, including the pointer used by CFS to locate the file (not displayed to the user), the file storage group being used, and whether the file is on disk or tape.
- The original system type (i.e., UNIX).
- Security and administrative information, including the date and time the node was created, last modified, written, and read; the userid(s) of the user(s) who created and last accessed the node; and the security partition (always green on the K-25 CFS) and classification level of the file (always unclassified on the K-25 CFS).
- An estimate of how frequently the file will be accessed (see the keyword **nuse=** in Chapter 11).
- The master user (i.e., owner's) validation entry, which consists of the following:
  - — The userid of the user who is the owner of the node.
  - — All the owner's access rights.
  - — A password, if one has been specified for the owner's access.
  - — An access rights modifier; **s** is the default.
- Access control information (i.e., user validation entries) about any other userids for which access rights have been specified (see the keywords **aval=**, **cval=**, and **dval=** in Chapter 11). Note: access rights are not generally specified for individual files. To determine the protection for a file, the user must examine and, if the access rights modifiers **a** and **o** are used, accumulate the protection for the root and any intermediate nodes.
- The file size in bits and whether the file is compressed. (Note that compression is not discussed in this manual; it is not available for files in UNICOS format.)
- The file release date (see the keyword **nrel=** in Chapter 11).
- The number of times the file has been accessed during the current accounting period.

## Appendix B
## Text File Format Conversion

You may have text files that were created on CTSS stored on CFS. Before using these files on UNICOS, you must convert them to UNICOS format. The conversion procedure is described below. For simplicity, the assumption is made that the CFS files are located directly under your userid root and that the files on UNICOS are located in your working directory (the one from which you invoked `cfsi`).

- Invoke `cfsi` on UNICOS.

  ```
  cfsi
  ```

- Copy your CTSS format file (*oldfile*) from CFS to the UNICOS Cray.

  **get** *oldfile*

  The file will have the same name on UNICOS as it has on CFS.

- Exit `cfsi`.

  ```
  end
  ```

- Convert the file to UNICOS format using the `ctou` program

  **ctou** *oldfile newfile*

  where *oldfile* is the current name of the file and *newfile* is the name you select for the converted version.

- Invoke `cfsi`.

  ```
  cfsi
  ```

- Save a copy of the converted file on CFS, retaining the file name.

  **save** *newfile*

- Delete the CTSS version of the file from CFS.

  **delete** *oldfile*

- Exit `cfsi`.

  ```
  end
  ```

- Delete the CTSS version of the file from your UNICOS area.

  **rm** *oldfile*

# Appendix C
# More About Specifying Workerfile Parameters

On UNICOS, the directory in which you are currently located is known as your *current working directory* or simply as your *working directory*. Use the command `ls` to list the contents of the directory and `pwd` to display the complete pathname for the directory. To change working directories, use `cd`.

When using `cfsi`, the UNICOS directory from which you invoked that program is your current working directory. For simplicity, the examples in this manual make two assumptions:

- when entering a `get` request, you want the CFS file copied to your current working directory; and
- when entering a `save`, `store`, or `replace` request, the file you want to save is located in your current working directory.

Under these circumstances, a workerfile parameter is required only if the file on UNICOS and CFS will have different names. In addition, the necessary workerfile specification is simply the name of the file on UNICOS. For example, the following request is all that is required to retrieve a copy of the file `trials` from CFS and place it in your UNICOS working directory as `trials`.

```
get trials
```

And, entering

```
get dryrun:trials
```

will give the copy of `trials` the name `dryrun` on UNICOS.

You can direct files to and from any of your UNICOS directories without exiting `cfsi`, changing working directories, and re-invoking `cfsi`. Two methods are available as described below.

- You can use the UNICOS `cd` command from within `cfsi` to change working directories. To move to a subdirectory of your working directory, indicate the appropriate relative pathname. To move to a parent or other higher-level directory, supply the full pathname (some shortcuts such as using `..` to represent the parent of your current working directory, are accepted, others are not). The file names on UNICOS and CFS may be the same or different.

  To illustrate, assume the full pathname for your home directory on UNICOS is /usr/u/000/*uid*, where *uid* represents your three-character UID. You have a tree structure that includes subdirectories at two levels; the respective full pathnames are /usr/u/000/*uid*/subdir1 and /usr/u/000/*uid*/subdir1/subdir2. You invoke cfsi from subdir1 (i.e., your current working directory is subdir1). Now, to retrieve a copy of the CFS file testfile and place it under subdir1, enter

```
get testfile
```

To place a copy of testfile under subdir2, enter

```
cd subdir2
get testfile
```

If you want the copy on UNICOS to have the name newfile, replace the last command with

```
get newfile:testfile
```

To place a copy of testfile in your home directory, enter

```
cd /usr/u/000/uid   or   cd ..
get testfile
```

In the last example, you cannot move to your home directory, as might be expected, by simply entering cd.

The UNICOS commands ls and pwd also work from within cfsi

When you end your cfsi session, you are always returned to the UNICOS directory from which you invoked the cfsi program.

- You can use a UNICOS pathname as the CFS workerfile parameter. If the UNICOS file will be copied to or from a subdirectory of your working directory, indicate the appropriate relative pathname of the file. If the UNICOS file will be copied to or from a parent or other higher-level directory, supply the full pathname for the file (some shortcuts such as using .. to represent the parent of your current working directory, are accepted, others are not). The file names on UNICOS and CFS may be the same or different.

  To illustrate, make the same assumptions as for the method just described: the full pathname for your home directory on UNICOS is /usr/u/000/*uid*, where *uid* represents your three-character UID; you have a tree structure that includes subdirectories with the full pathnames /usr/u/000/*uid*/subdir1 and /usr/u/000/*uid*/subdir1/subdir2; and you invoke cfsi from subdir1 (i.e., your current working directory is subdir1). Now, to retrieve a copy of the CFS file testfile and place it under subdir1, enter

  ```
  get testfile
  ```

To place a copy of **testfile** under **subdir2**, enter

```
get subdir2/testfile:testfile
```

To place a copy of **testfile** under **subdir2** and call the copy on UNICOS **newfile**, enter

```
get subdir2/newfile:testfile
```

To place a copy of **testfile** in your home directory, enter

```
get /usr/u/000/uid/testfile:testfile
```

or

```
get ../testfile:testfile
```

## Appendix D
## Protection Mode for CFS Files Copied to UNICOS

Access permissions on UNICOS can be represented in symbolic mode (i.e., using the letters *r*, *w*, and *x* to represent read, write, and execute) or as absolute values given in octal notation (e.g., 777 represents full access for all users). For information about protection modes on UNICOS, see the handout *Converting to UNICOS*, which is available from Programming Assistance.

In general, UNICOS assigns a default protection mode to each file created; the mode depends on how the file was created. For example, the default mode for an object file produced by a compiler is 666. The system defaults can be modified using the **umask** command to establish a user mask. The argument for the command is the octal number that, when subtracted from the system defaults, gives the desired protection modes. For example, on the STC UNICOS Cray, the command

```
umask 022
```

has been placed in the system initialization file. This mask causes object files to be created with the protection mode 644 instead of 666; the protection for files created in other ways is similarly modified. If you want, you can place a more restrictive **umask** command in your personal initialization file; this command will override the system value.

The protection mode for files you retrieve from CFS, although based on the user mask in effect for your session, is determined by **cfsi**. The **cfsi** program assigns the protection mode that results from subtracting your user mask from 777. Unless you have placed a different **umask** command in your personal initialization file, this user mask is 022, as described above. As a result, the protection mode for files you retrieve using the **cfsi get** command will be 755 or

| | |
|---|---|
| Owner | read, write, execute |
| Group | read, execute |
| World | read, execute |

At a minimum, **cfsi** will always grant read permission to the owner.

# Index

## CREDITS

*Using the K-25 C&TD Common File System: A Guide to CFSI* was prepared by Sandra Edwards with technical assistance from Sandy Guinn, John McMillan, Debbie Bryant, Janice Hensley, Mark Smith, and Kris Norris. Diagrams were produced by Sherry Gordon.

*Using the K-25 C&TD Common File System: A Guide to CFSI* is a local adaptation of the *Common File System CFS Interface Reference*, a manual produced by the Los Alamos National Laboratory (prepared by Ted Spitzmiller with technical assistance from Tyce McLarty, Marge Devany, Catherine Mexal, and Glen Carter). That manual bears the following notice: