

[About](#)
▼[Get
FreeBSD](#)
▼[Documentation](#)
▼[Community](#)
▼[Donate](#)[☰ Book menu](#)

Chapter 3. FreeBSD Basics

Table of Contents

- 3.1. Synopsis
- 3.2. Virtual Consoles and Terminals
- 3.3. Users and Basic Account Management
- 3.4. Permissions
- 3.5. Directory Structure
- 3.6. Disk Organization
- 3.7. Mounting and Unmounting File Systems
- 3.8. Processes and Daemons
- 3.9. Shells
- 3.10. Text Editors
- 3.11. Devices and Device Nodes
- 3.12. Manual Pages

3.1. Synopsis

This chapter covers the basic commands and functionality of the FreeBSD operating system. Much of this material is relevant for any UNIX®-like operating system. New FreeBSD users are encouraged to read through this chapter carefully.

After reading this chapter, you will know:

- How to use and configure virtual consoles.
- How to create and manage users and groups on FreeBSD.
- How UNIX® file permissions and FreeBSD file flags work.



- The default FreeBSD file system layout.
- The FreeBSD disk organization.
- How to mount and unmount file systems.
- What processes, daemons, and signals are.
- What a shell is, and how to change the default login environment.
- How to use basic text editors.
- What devices and device nodes are.
- How to read manual pages for more information.

3.2. Virtual Consoles and Terminals

Unless FreeBSD has been configured to automatically start a graphical environment during startup, the system will boot into a command line login prompt, as seen in this example:

```
FreeBSD/amd64 (pc3.example.org) (ttyv0)
```

```
login:
```

The first line contains some information about the system. The `amd64` indicates that FreeBSD is running on a 64-bit x86 system. The hostname is `pc3.example.org`, and `ttyv0` indicates that this is the "system console". The second line is the login prompt.

Since FreeBSD is a multiuser system, it needs some way to distinguish between different users. This is accomplished by requiring every user to log into the system before gaining access to the programs on the system. Every user has a unique "username" and a personal "password".

To log into the system console, type the username that was configured during system installation, as described in [Add Users](#), and press . Then enter the password associated with the username and press . The password is *not echoed* for security reasons.

Once the correct password is input, the message of the day (MOTD) will be displayed followed by a command prompt. Depending upon the shell that was selected when the



user was created, this prompt will be a `#`, `$`, or `%` character. The prompt indicates that the user is now logged into the FreeBSD system console and ready to try the available commands.

3.2.1. Virtual Consoles

While the system console can be used to interact with the system, a user working from the command line at the keyboard of a FreeBSD system will typically instead log into a virtual console. This is because system messages are configured by default to display on the system console. These messages will appear over the command or file that the user is working on, making it difficult to concentrate on the work at hand.

By default, FreeBSD is configured to provide several virtual consoles for inputting commands. Each virtual console has its own login prompt and shell and it is easy to switch between virtual consoles. This essentially provides the command line equivalent of having several windows open at the same time in a graphical environment.

The key combinations `Alt + F1` through `Alt + F8` have been reserved by FreeBSD for switching between virtual consoles. Use `Alt + F1` to switch to the system console (`ttyv0`), `Alt + F2` to access the first virtual console (`ttyv1`), `Alt + F3` to access the second virtual console (`ttyv2`), and so on. When using Xorg as a graphical console, the combination becomes `Ctrl + Alt + F1` to return to a text-based virtual console.

When switching from one console to the next, FreeBSD manages the screen output. The result is an illusion of having multiple virtual screens and keyboards that can be used to type commands for FreeBSD to run. The programs that are launched in one virtual console do not stop running when the user switches to a different virtual console.

Refer to [kbdcontrol\(1\)](#), [vidcontrol\(1\)](#), [atkbd\(4\)](#), [syscons\(4\)](#), and [vt\(4\)](#) for a more technical description of the FreeBSD console and its keyboard drivers.

In FreeBSD, the number of available virtual consoles is configured in this section of `/etc/ttys`:

#	name	getty	type	status	comments
#	ttyv0	"/usr/libexec/getty Pc"	xterm	on	secure
#	Virtual terminals				
	ttyv1	"/usr/libexec/getty Pc"	xterm	on	secure
	ttyv2	"/usr/libexec/getty Pc"	xterm	on	secure
	ttyv3	"/usr/libexec/getty Pc"	xterm	on	secure



```

ttyv4    "/usr/libexec/getty Pc"         xterm  on  secure
ttyv5    "/usr/libexec/getty Pc"         xterm  on  secure
ttyv6    "/usr/libexec/getty Pc"         xterm  on  secure
ttyv7    "/usr/libexec/getty Pc"         xterm  on  secure
ttyv8    "/usr/X11R6/bin/xdm -nodaemon"  xterm  off secure

```

To disable a virtual console, put a comment symbol (`#`) at the beginning of the line representing that virtual console. For example, to reduce the number of available virtual consoles from eight to four, put a `#` in front of the last four lines representing virtual consoles `ttyv5` through `ttyv8`. *Do not* comment out the line for the system console `ttyv0`. Note that the last virtual console (`ttyv8`) is used to access the graphical environment if Xorg has been installed and configured as described in [The X Window System](#).

For a detailed description of every column in this file and the available options for the virtual consoles, refer to [ttyps\(5\)](#).

3.2.2. Single User Mode

The FreeBSD boot menu provides an option labelled as "Boot Single User". If this option is selected, the system will boot into a special mode known as "single user mode". This mode is typically used to repair a system that will not boot or to reset the `root` password when it is not known. While in single user mode, networking and other virtual consoles are not available. However, full `root` access to the system is available, and by default, the `root` password is not needed. For these reasons, physical access to the keyboard is needed to boot into this mode and determining who has physical access to the keyboard is something to consider when securing a FreeBSD system.

The settings which control single user mode are found in this section of `/etc/ttys`:

```

# name  getty                                type  status  comments
#
# If console is marked "insecure", then init will ask for the root p
# when going to single-user mode.
console none                                unknown off  secure

```

By default, the status is set to `secure`. This assumes that who has physical access to the keyboard is either not important or it is controlled by a physical security policy. If this setting is changed to `insecure`, the assumption is that the environment itself is



insecure because anyone can access the keyboard. When this line is changed to `insecure`, FreeBSD will prompt for the `root` password when a user selects to boot into single user mode.

Note

Be careful when changing this setting to `insecure`! If the `root` password is forgotten, booting into single user mode is still possible, but may be difficult for someone who is not familiar with the FreeBSD booting process.

3.2.3. Changing Console Video Modes

The FreeBSD console default video mode may be adjusted to 1024×768, 1280×1024, or any other size supported by the graphics chip and monitor. To use a different video mode load the `VESA` module:

```
# kldload vesa
```



To determine which video modes are supported by the hardware, use [vidcontrol\(1\)](#). To get a list of supported video modes issue the following:

```
# vidcontrol -i mode
```



The output of this command lists the video modes that are supported by the hardware. To select a new video mode, specify the mode using [vidcontrol\(1\)](#) as the `root` user:

```
# vidcontrol MODE_279
```



If the new video mode is acceptable, it can be permanently set on boot by adding it to `/etc/rc.conf`:

```
allscreens_flags="MODE_279"
```



3.3. Users and Basic Account Management

FreeBSD allows multiple users to use the computer at the same time. While only one user can sit in front of the screen and use the keyboard at any one time, any number of users can log in to the system through the network. To use the system, each user should have their own user account.

This chapter describes:

- The different types of user accounts on a FreeBSD system.
- How to add, remove, and modify user accounts.
- How to set limits to control the resources that users and groups are allowed to access.
- How to create groups and add users as members of a group.

3.3.1. Account Types

Since all access to the FreeBSD system is achieved using accounts and all processes are run by users, user and account management is important.

There are three main types of accounts: system accounts, user accounts, and the superuser account.


3.3.1.1. System Accounts

System accounts are used to run services such as DNS, mail, and web servers. The reason for this is security; if all services ran as the superuser, they could act without restriction.

Examples of system accounts are `daemon`, `operator`, `bind`, `news`, and `www`.

`nobody` is the generic unprivileged system account. However, the more services that use `nobody`, the more files and processes that user will become associated with, and hence the more privileged that user becomes.

3.3.1.2. User Accounts

User accounts are assigned to real people and are used to log in and use the system.  Every person accessing the system should have a unique user account. This allows the

administrator to find out who is doing what and prevents users from clobbering the settings of other users.

Each user can set up their own environment to accommodate their use of the system, by configuring their default shell, editor, key bindings, and language settings.

Every user account on a FreeBSD system has certain information associated with it:

User name

The user name is typed at the `login:` prompt. Each user must have a unique user name. There are a number of rules for creating valid user names which are documented in [passwd\(5\)](#). It is recommended to use user names that consist of eight or fewer, all lower case characters in order to maintain backwards compatibility with applications.

Password

Each account has an associated password.

User ID (UID)

The User ID (UID) is a number used to uniquely identify the user to the FreeBSD system. Commands that allow a user name to be specified will first convert it to the UID. It is recommended to use a UID less than 65535, since higher values may cause compatibility issues with some software.

Group ID (GID)

The Group ID (GID) is a number used to uniquely identify the primary group that the user belongs to. Groups are a mechanism for controlling access to resources based on a user's GID rather than their UID. This can significantly reduce the size of some configuration files and allows users to be members of more than one group. It is recommended to use a GID of 65535 or lower as higher GIDs may break some software.

Login class

Login classes are an extension to the group mechanism that provide additional flexibility when tailoring the system to different users. Login classes are discussed further in [Configuring Login Classes](#).

Password change time



By default, passwords do not expire. However, password expiration can be enabled on a per-user basis, forcing some or all users to change their passwords after a certain amount of time has elapsed.

Account expiration time

By default, FreeBSD does not expire accounts. When creating accounts that need a limited lifespan, such as student accounts in a school, specify the account expiry date using `pw(8)`. After the expiry time has elapsed, the account cannot be used to log in to the system, although the account's directories and files will remain.

User's full name

The user name uniquely identifies the account to FreeBSD, but does not necessarily reflect the user's real name. Similar to a comment, this information can contain spaces, uppercase characters, and be more than 8 characters long.

Home directory

The home directory is the full path to a directory on the system. This is the user's starting directory when the user logs in. A common convention is to put all user home directories under `/home/username` or `/usr/home/username`. Each user stores their personal files and subdirectories in their own home directory.

User shell

The shell provides the user's default environment for interacting with the system. There are many different kinds of shells and experienced users will have their own preferences, which can be reflected in their account settings.

3.3.1.3. The Superuser Account

The superuser account, usually called `root`, is used to manage the system with no limitations on privileges. For this reason, it should not be used for day-to-day tasks like sending and receiving mail, general exploration of the system, or programming.

The superuser, unlike other user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the operating system by mistake, so it is recommended to login as a user account and to only become the superuser when a command requires extra privilege.

Always double and triple-check any commands issued as the superuser, since an extra space or missing character can mean irreparable data loss.



There are several ways to gain superuser privilege. While one can log in as `root`, this is highly discouraged.

Instead, use `su(1)` to become the superuser. If `-` is specified when running this command, the user will also inherit the root user's environment. The user running this command must be in the `wheel` group or else the command will fail. The user must also know the password for the `root` user account.

In this example, the user only becomes superuser in order to run `make install` as this step requires superuser privilege. Once the command completes, the user types `exit` to leave the superuser account and return to the privilege of their user account.

Example 1. Install a Program As the Superuser



```
% configure
% make
% su -
Password:
# make install
# exit
%
```

The built-in `su(1)` framework works well for single systems or small networks with just one system administrator. An alternative is to install the `security/sudo` package or port. This software provides activity logging and allows the administrator to configure which users can run which commands as the superuser.

3.3.2. Managing Accounts

FreeBSD provides a variety of different commands to manage user accounts. The most common commands are summarized in [Utilities for Managing User Accounts](#), followed by some examples of their usage. See the manual page for each utility for more details and usage examples.

Table 1. Utilities for Managing User Accounts

Command	Summary
---------	---------



adduser(8)

The recommended command-line application for adding new users.

rmuser(8)

The recommended command-line application for removing users.

chpass(1)

A flexible tool for changing user database information.

passwd(1)

The command-line tool to change user passwords.

pw(8)

A powerful and flexible tool for modifying all aspects of user accounts.

bsdconfig(8)

A system configuration utility with account management support.

3.3.2.1. Adding a user

The recommended program for adding new users is [adduser\(8\)](#). When a new user is added, this program automatically updates `/etc/passwd` and `/etc/group`. It also creates a home directory for the new user, copies in the default configuration files from `/usr/share/skel`, and can optionally mail the new user a welcome message. This utility must be run as the superuser.

The [adduser\(8\)](#) utility is interactive and walks through the steps for creating a new user account. As seen in [Adding a User on FreeBSD](#), either input the required information or press to accept the default value shown in square brackets. In this example, the user has been invited into the `wheel` group, allowing them to become the superuser with [su\(1\)](#). When finished, the utility will prompt to either create another user or to exit.

Example 2. Adding a User on FreeBSD





```
# adduser
```

The output should be similar to the following:

```
Username: jru
Full name: J. Random User
Uid (Leave empty for default):
Login group [jru]:
Login group is jru. Invite jru into other groups? []: wheel
Login class [default]:
Shell (sh csh tcsh zsh nologin) [sh]: zsh
Home directory [/home/jru]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username      : jru
Password      : ****
Full Name     : J. Random User
Uid           : 1001
Class         :
Groups        : jru wheel
Home          : /home/jru
Shell         : /usr/local/bin/zsh
Locked        : no
OK? (yes/no): yes
adduser: INFO: Successfully added (jru) to the user database.
Add another user? (yes/no): no
Goodbye!
```



Note

Since the password is not echoed when typed, be careful to not mistype the password when creating the user account.

3.3.2.2. Removing a user

To completely remove a user from the system, run `rmuser(8)` as the superuser. This command performs the following steps:

1. Removes the user's `crontab(1)` entry, if one exists.
2. Removes any `at(1)` jobs belonging to the user.
3. Sends a SIGKILL signal to all processes owned by the user.
4. Removes the user from the system's local password file.
5. Removes the user's home directory (if it is owned by the user), including handling of symbolic links in the path to the actual home directory.
6. Removes the incoming mail files belonging to the user from `/var/mail`.
7. Removes all files owned by the user from `/tmp`, `/var/tmp`, and `/var/tmp/vi.recover`.
8. Removes the username from all groups to which it belongs in `/etc/group`. (If a group becomes empty and the group name is the same as the username, the group is removed; this complements `adduser(8)`'s per-user unique groups.)
9. Removes all message queues, shared memory segments and semaphores owned by the user.

`rmuser(8)` cannot be used to remove superuser accounts since that is almost always an indication of massive destruction.

By default, an interactive mode is used, as shown in the following example.

Example 3. `rmuser` Interactive Account Removal



```
# rmuser jru
```



The output should be similar to the following:

```
Matching password entry:
jru:*:1001:1001::0:0:J. Random User:/home/jru:/usr/local/bin/zsh
Is this the entry you wish to remove? y
Remove user's home directory (/home/jru)? y
Removing user (jru): mailspool home passwd.
```

3.3.2.3. Change user information

Any user can use [chpass\(1\)](#) to change their default shell and personal information associated with their user account. The superuser can use this utility to change additional account information for any user.

When passed no options, aside from an optional username, [chpass\(1\)](#) displays an editor containing user information. When the user exits from the editor, the user database is updated with the new information.

Note

This utility will prompt for the user's password when exiting the editor, unless the utility is run as the superuser.

In [Using chpass as Superuser](#), the superuser has typed `chpass jru` and is now viewing the fields that can be changed for this user. If `jru` runs this command instead, only the last six fields will be displayed and available for editing. This is shown in [Using chpass as Regular User](#).

Example 4. Using chpass as Superuser


```
# chpass jru
```



The output should be similar to the following:

```
# Changing user database information for jru.  
Login: jru  
Password: *  
Uid [#]: 1001  
Gid [# or name]: 1001  
Change [month day year]:  
Expire [month day year]:  
Class:  
Home directory: /home/jru  
Shell: /usr/local/bin/zsh  
Full Name: J. Random User  
Office Location:  
Office Phone:  
Home Phone:  
Other information:
```

Example 5. Using `chpass` as Regular User



```
#Changing user database information for jru.  
Shell: /usr/local/bin/zsh  
Full Name: J. Random User  
Office Location:  
Office Phone:  
Home Phone:  
Other information:
```

Note

The commands [chfn\(1\)](#) and [chsh\(1\)](#) are links to [chpass\(1\)](#), as are [ypchpass\(1\)](#), [ypchfn\(1\)](#), and [ypchsh\(1\)](#). Since NIS support is automatic, specifying the `yp` before the command is not necessary. How to configure NIS is covered in [Network Servers](#).



3.3.2.4. Change user password

Any user can easily change their password using [passwd\(1\)](#). To prevent accidental or unauthorized changes, this command will prompt for the user's original password before a new password can be set:

Example 6. Changing Your Password

```
% passwd
```



The output should be similar to the following:

```
Changing local password for jru.  
Old password:  
New password:  
Retype new password:  
passwd: updating the database...  
passwd: done
```

The superuser can change any user's password by specifying the username when running [passwd\(1\)](#). When this utility is run as the superuser, it will not prompt for the user's current password. This allows the password to be changed when a user cannot remember the original password.

Example 7. Changing Another User's Password as the Superuser

```
# passwd jru
```



The output should be similar to the following:



```
Changing local password for jru.  
New password:  
Retype new password:  
passwd: updating the database...  
passwd: done
```

Note

As with [chpass\(1\)](#), [yppasswd\(1\)](#) is a link to [passwd\(1\)](#), so NIS works with either command.

3.3.2.5. Create, remove, modify and display system users and groups

The [pw\(8\)](#) utility can create, remove, modify, and display users and groups. It functions as a front end to the system user and group files. [pw\(8\)](#) has a very powerful set of command line options that make it suitable for use in shell scripts, but new users may find it more complicated than the other commands presented in this section.

3.3.3. Managing Groups

A group is a list of users. A group is identified by its group name and GID. In FreeBSD, the kernel uses the UID of a process, and the list of groups it belongs to, to determine what the process is allowed to do. Most of the time, the GID of a user or process usually means the first group in the list.

The group name to GID mapping is listed in `/etc/group`. This is a plain text file with four colon-delimited fields. The first field is the group name, the second is the encrypted password, the third the GID, and the fourth the comma-delimited list of members. For a complete description of the syntax, refer to [group\(5\)](#).

The superuser can modify `/etc/group` using a text editor, although editing the group file using [vigr\(8\)](#) is preferred because it can catch some common mistakes. Alternatively, [pw\(8\)](#) can be used to add and edit groups. For example, to add a group called `teamtwo` and then confirm that it exists:



Warning

Care must be taken when using the operator group, as unintended superuser-like access privileges may be granted, including but not limited to shutdown, reboot, and access to all items in `/dev` in the group.

Example 8. Adding a Group Using `pw(8)`



```
# pw groupadd teamtwo
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:
```

In this example, 1100 is the GID of `teamtwo`. Right now, `teamtwo` has no members. This command will add `jru` as a member of `teamtwo`.

Example 9. Adding User Accounts to a New Group Using `pw(8)`



```
# pw groupmod teamtwo -M jru
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:jru
```

The argument to `-M` is a comma-delimited list of users to be added to a new (empty) group or to replace the members of an existing group. To the user, this group membership is different from (and in addition to) the user's primary group listed in the password file. This means that the user will not show up as a member when using `groupshow` with `pw(8)`, but will show up when the information is queried via `id(1)` or a



similar tool. When [pw\(8\)](#) is used to add a user to a group, it only manipulates `/etc/group` and does not attempt to read additional data from `/etc/passwd`.

Example 10. Adding a New Member to a Group Using [pw\(8\)](#)



```
# pw groupmod teamtwo -m db
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:jru,db
```

In this example, the argument to `-m` is a comma-delimited list of users who are to be added to the group. Unlike the previous example, these users are appended to the group and do not replace existing users in the group.

Example 11. Using [id\(1\)](#) to Determine Group Membership



```
% id jru
```

The output should be similar to the following:

```
uid=1001(jru) gid=1001(jru) groups=1001(jru), 1100(teamtwo)
```

In this example, `jru` is a member of the groups `jru` and `teamtwo`.

For more information about this command and the format of `/etc/group`, refer to [pw\(8\)](#) and [group\(5\)](#).

3.4. Permissions

In FreeBSD, every file and directory has an associated set of permissions and several utilities are available for viewing and modifying these permissions. Understanding how



permissions work is necessary to make sure that users are able to access the files that they need and are unable to improperly access the files used by the operating system or owned by other users.

This section discusses the traditional UNIX® permissions used in FreeBSD. For finer-grained file system access control, refer to [Access Control Lists](#).

In UNIX®, basic permissions are assigned using three types of access: read, write, and execute. These access types are used to determine file access to the file's owner, group, and others (everyone else). The read, write, and execute permissions can be represented as the letters `r`, `w`, and `x`. They can also be represented as binary numbers as each permission is either on or off (0). When represented as a number, the order is always read as `rwX`, where `r` has an on value of 4, `w` has an on value of 2 and `x` has an on value of 1.

Table 4.1 summarizes the possible numeric and alphabetic possibilities. When reading the "Directory Listing" column, a `-` is used to represent a permission that is set to off.

Table 2. UNIX® Permissions

Value	Permission	Directory Listing
0	No read, no write, no execute	- - -
1	No read, no write, execute	- - x
2	No read, write, no execute	- w -
3	No read, write, execute	- w x
4	Read, no write, no execute	r - -
5	Read, no write, execute	r - x



Value	Permission	Directory Listing
6	Read, write, no execute	rw-
7	Read, write, execute	rwx

Use the `-l` argument with [ls\(1\)](#) to view a long directory listing that includes a column of information about a file's permissions for the owner, group, and everyone else. For example, `ls -l` in an arbitrary directory may show:

```
% ls -l
```



The output should be similar to the following:

```
total 530
-rw-r--r--  1 root  wheel    512 Sep  5 12:31 myfile
-rw-r--r--  1 root  wheel    512 Sep  5 12:31 otherfile
-rw-r--r--  1 root  wheel   7680 Sep  5 12:31 email.txt
```

Focusing on the line for `myfile`, the first (leftmost) character indicates whether this file is a regular file, a directory, a special character device, a socket, or any other special pseudo-file device. In this example, the `-` indicates a regular file. The next three characters, `rw-` in this example, give the permissions for the owner of the file. The next three characters, `r--`, give the permissions for the group that the file belongs to. The final three characters, `r--`, give the permissions for the rest of the world. A dash means that the permission is turned off. In this example, the permissions are set so the owner can read and write to the file, the group can read the file, and the rest of the world can only read the file. According to the table above, the permissions for this file would be `644`, where each digit represents the three parts of the file's permission.

How does the system control permissions on devices? FreeBSD treats most hardware devices as a file that programs can open, read, and write data to. These special device files are stored in `/dev/`.

Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a



directory is marked executable, it means it is possible to change into that directory using [cd\(1\)](#). This also means that it is possible to access the files within that directory, subject to the permissions on the files themselves.

In order to perform a directory listing, the read permission must be set on the directory. In order to delete a file that one knows the name of, it is necessary to have write *and* execute permissions to the directory containing the file.

There are more permission bits, but they are primarily used in special circumstances such as setuid binaries and sticky directories. For more information on file permissions and how to set them, refer to [chmod\(1\)](#).

3.4.1. Symbolic Permissions

Symbolic permissions use characters instead of octal values to assign permissions to files or directories. Symbolic permissions use the syntax of (who) (action) (permissions), where the following values are available:

Option	Letter	Represents
(who)	u	User
(who)	g	Group owner
(who)	o	Other
(who)	a	All ("world")
(action)	+	Adding permissions
(action)	-	Removing permissions



Option	Letter	Represents
(action)	=	Explicitly set permissions
(permissions)	r	Read
(permissions)	w	Write
(permissions)	x	Execute
(permissions)	t	Sticky bit
(permissions)	s	Set UID or GID

These values are used with [chmod\(1\)](#), but with letters instead of numbers. For example, the following command would block both members of the group associated with *FILE* and all other users from accessing *FILE*:

```
% chmod go= FILE
```



A comma separated list can be provided when more than one set of changes to a file must be made. For example, the following command removes the group and "world" write permission on *FILE*, and adds the execute permissions for everyone:

```
% chmod go-w,a+x FILE
```



3.4.2. FreeBSD File Flags



In addition to file permissions, FreeBSD supports the use of "file flags". These flags add

an additional level of security and control over files, but not directories. With file flags, even `root` can be prevented from removing or altering files.

File flags are modified using [chflags\(1\)](#). For example, to enable the system undeletable flag on the file `file1`, issue the following command:

```
# chflags sunlink file1
```



To disable the system undeletable flag, put a "no" in front of the `sunlink`:

```
# chflags nosunlink file1
```



To view the flags of a file, use `-lo` with [ls\(1\)](#):

```
# ls -lo file1
```



```
-rw-r--r--  1 trhodes  trhodes  sunlnk 0 Mar  1 05:54 file1
```

Several file flags may only be added or removed by the `root` user. In other cases, the file owner may set its file flags. Refer to [chflags\(1\)](#) and [chflags\(2\)](#) for more information.

3.4.3. The `setuid`, `setgid`, and sticky Permissions

Other than the permissions already discussed, there are three other specific settings that all administrators should know about. They are the `setuid`, `setgid`, and sticky permissions.

These settings are important for some UNIX® operations as they provide functionality not normally granted to normal users. To understand them, the difference between the real user ID and effective user ID must be noted.

The real user ID is the UID who owns or starts the process. The effective UID is the user ID the process runs as. As an example, [passwd\(1\)](#) runs with the real user ID when a user changes their password. However, in order to update the password database, the command runs as the effective ID of the `root` user. This allows users to change their



passwords without seeing a `Permission Denied` error.

The `setuid` permission may be added symbolically by adding the `s` permission for the user as in the following example:

```
# chmod u+s suidexample.sh
```



The `setuid` permission may also be set by prefixing a permission set with the number four (4) as shown in the following example:

```
# chmod 4755 suidexample.sh
```



The permissions on `suidexample.sh` now look like the following:

```
-rwsr-xr-x  1 trhodes  trhodes   63 Aug 29 06:36 suidexample.sh
```

Note that a `s` is now part of the permission set designated for the file owner, replacing the executable bit. This allows utilities which need elevated permissions, such as [passwd\(1\)](#).

Note

The `nosuid` [mount\(8\)](#) option will cause such binaries to silently fail without alerting the user. That option is not completely reliable as a `nosuid` wrapper may be able to circumvent it.

To view this in real time, open two terminals. On one, type `passwd` as a normal user. While it waits for a new password, check the process table and look at the user information for [passwd\(1\)](#):

In terminal A:

```
Changing local password for trhodes
Old Password:
```



In terminal B:

```
# ps aux | grep passwd
```

```
trhodes  5232  0.0  0.2  3420  1608   0  R+   2:10AM   0:00.00 grep passwd
root      5211  0.0  0.2  3620  1724   2  I+   2:09AM   0:00.01 passwd
```

Although `passwd(1)` is run as a normal user, it is using the effective UID of `root`.

The `setgid` permission performs the same function as the `setuid` permission; except that it alters the group settings. When an application or utility executes with this setting, it will be granted the permissions based on the group that owns the file, not the user who started the process.

To set the `setgid` permission on a file symbolically, add the `s` permission for the group with `chmod(1)`:

```
# chmod g+s sgidexample.sh
```

Alternatively, provide `chmod(1)` with a leading two (2):

```
# chmod 2755 sgidexample.sh
```

In the following listing, notice that the `s` is now in the field designated for the group permission settings:

```
-rwxr-sr-x  1 trhodes  trhodes   44 Aug 31 01:49 sgidexample.sh
```

Note

In these examples, even though the shell script in question is an executable file, it will not run with a different EUID or effective user ID. This is because shell scripts may not access the `setuid(2)` system calls.

The `setuid` and `setgid` permission bits may lower system security, by allowing for elevated permissions. The third special permission, the `sticky bit`, can strengthen the security of a system.

When the `sticky bit` is set on a directory, it allows file deletion only by the file owner. This is useful to prevent file deletion in public directories, such as `/tmp`, by users who do not own the file. To utilize this permission, add the `t` mode to the file:

```
# chmod +t /tmp
```



Alternatively, prefix the permission set with a one (1):

```
# chmod 1777 /tmp
```



The `sticky bit` permission will display as a `t` at the very end of the permission set:

```
# ls -al / | grep tmp
```



```
drwxrwxrwt  10 root  wheel           512 Aug 31 01:49 tmp
```



3.5. Directory Structure

The FreeBSD directory hierarchy is fundamental to obtaining an overall understanding of the system. The most important directory is root or, `/`. This directory is the first one mounted at boot time and it contains the base system necessary to prepare the operating system for multi-user operation. The root directory also contains mount points for other file systems that are mounted during the transition to multi-user operation.



A mount point is a directory where additional file systems can be grafted onto a parent file system (usually the root file system). This is further described in [Disk Organization](#). Standard mount points include `/usr/`, `/var/`, `/tmp/`, `/mnt/`, and `/cdrom/`. These directories are usually referenced to entries in `/etc/fstab`. This file is a table of various file systems and mount points and is read by the system. Most of the file systems in `/etc/fstab` are mounted automatically at boot time from the script [rc\(8\)](#) unless their entry includes `noauto`. Details can be found in [The fstab File](#).

A complete description of the file system hierarchy is available in [hier\(7\)](#). The following table provides a brief overview of the most common directories.

Directory	Description
<code>/</code>	Root directory of the file system.
<code>/bin/</code>	User utilities fundamental to both single-user and multi-user environments.
<code>/boot/</code>	Programs and configuration files used during operating system bootstrap.
<code>/boot/defaults/</code>	Default boot configuration files. Refer to loader.conf(5) for details.
<code>/dev/</code>	Device special files managed by devfs(5)
<code>/etc/</code>	System configuration files and scripts.
<code>/etc/defaults/</code>	Default system configuration files. Refer to rc(8) for details.



/etc/periodic/	Scripts that run daily, weekly, and monthly, via cron(8) . Refer to periodic(8) for details.
/lib/	Critical system libraries needed for binaries in <code>/bin</code> and <code>/sbin</code>
/libexec/	Critical system files
/media/	Contains subdirectories to be used as mount points for removable media such as CDs, USB drives, and floppy disks
/mnt/	Empty directory commonly used by system administrators as a temporary mount point.
/net/	Automounted NFS shares; see auto_master(5)
/proc/	Process file system. Refer to procfs(5) for details.
/rescue/	Statically linked programs for emergency recovery as described in rescue(8) .
/root/	Home directory for the <code>root</code> account.
/sbin/	System programs and administration utilities fundamental to both single-user and multi-user environments.



/tmp/	Temporary files which are usually <i>not</i> preserved across a system reboot. A memory-based file system is often mounted at <code>/tmp</code> . This can be automated using the <code>tmpmfs</code> -related variables of rc.conf(5) or with an entry in <code>/etc/fstab</code> ; refer to mdmfs(8) for details.
/usr/	The majority of user utilities and applications.
/usr/bin/	Common utilities, programming tools, and applications.
/usr/include/	Standard C include files.
/usr/lib/	Archive libraries.
/usr/libdata/	Miscellaneous utility data files.
/usr/libexec/	System daemons and system utilities executed by other programs.
/usr/local/	Local executables and libraries. Also used as the default destination for the FreeBSD ports framework. Within <code>/usr/local</code> , the general layout sketched out by hier(7) for <code>/usr</code> should be used. Exceptions are the <code>man</code> directory, which is directly under <code>/usr/local</code> rather than under <code>/usr/local/share</code> , and the ports documentation is in <code>share/doc/port</code> .
/usr/ports/	The FreeBSD Ports Collection (optional).



/usr/sbin/	System daemons and system utilities executed by users.
<hr/>	
/usr/share/	Architecture-independent files.
<hr/>	
/usr/src/	BSD and/or local source files.
<hr/>	
/var/	Multi-purpose log, temporary, transient, and spool files.
<hr/>	
/var/log/	Miscellaneous system log files.
<hr/>	
/var/tmp/	Temporary files which are usually preserved across a system reboot.
<hr/>	

3.6. Disk Organization

The smallest unit of organization that FreeBSD uses to find files is the filename. Filenames are case-sensitive, which means that `readme.txt` and `README.TXT` are two separate files. FreeBSD does not use the extension of a file to determine whether the file is a program, document, or some other form of data.

Files are stored in directories. A directory may contain no files, or it may contain many hundreds of files. A directory can also contain other directories, allowing a hierarchy of directories within one another in order to organize data.

Files and directories are referenced by giving the file or directory name, followed by a forward slash, `/`, followed by any other directory names that are necessary. For example, if the directory `foo` contains a directory `bar` which contains the file `readme.txt`, the full name, or *path*, to the file is `foo/bar/readme.txt`. Note that this is different from Windows® which uses `\` to separate file and directory names. FreeBSD does not use drive letters, or other drive names in the path. For example, one would not type `c:\foo\bar\readme.txt` on FreeBSD.

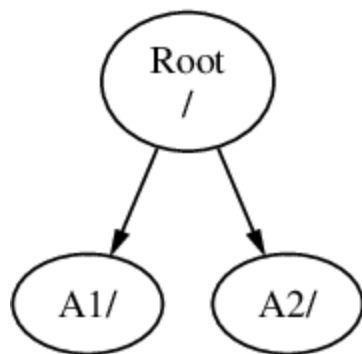


3.6.1. File systems

Directories and files are stored in a file system. Each file system contains exactly one directory at the very top level, called the *root directory* for that file system. This root directory can contain other directories. One file system is designated the *root file system* or */*. Every other file system is *mounted* under the root file system. No matter how many disks are on the FreeBSD system, every directory appears to be part of the same disk.

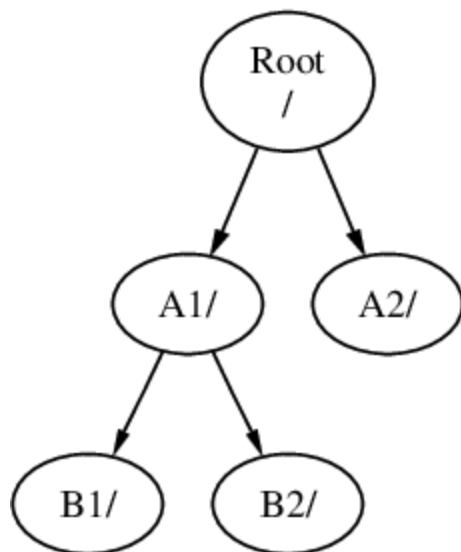
Consider three file systems, called A, B, and C. Each file system has one root directory, which contains two other directories, called A1, A2 (and likewise B1, B2 and C1, C2).

Call A the root file system. If `ls(1)` is used to view the contents of this directory, it will show two subdirectories, A1 and A2. The directory tree looks like this:



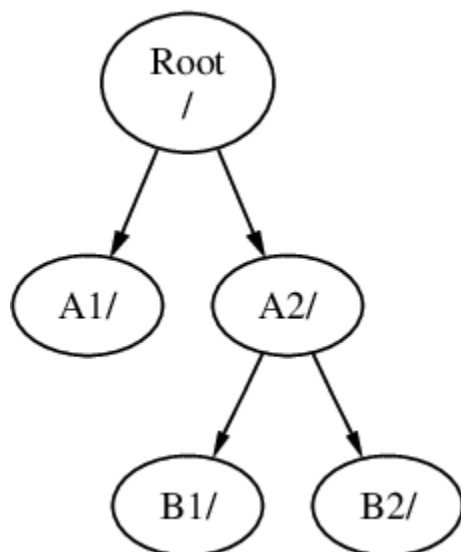
A file system must be mounted on to a directory in another file system. When mounting file system B on to the directory A1, the root directory of B replaces A1, and the directories in B appear accordingly:





Any files that are in the B1 or B2 directories can be reached with the path `/A1/B1` or `/A1/B2` as necessary. Any files that were in `/A1` have been temporarily hidden. They will reappear if B is *unmounted* from A.

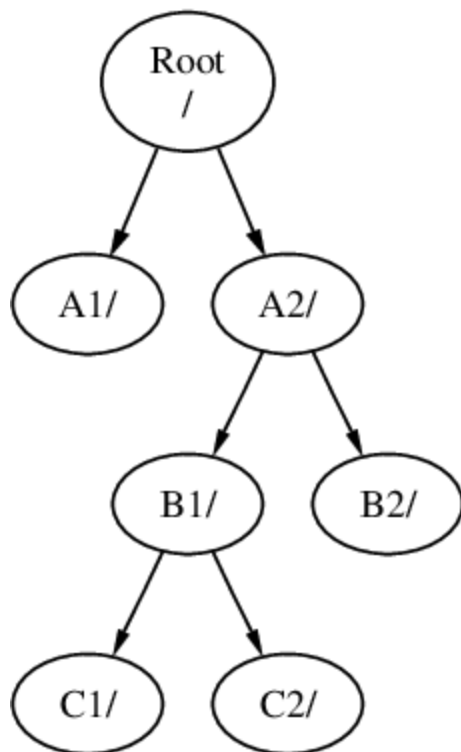
If B had been mounted on A2 then the diagram would look like this:



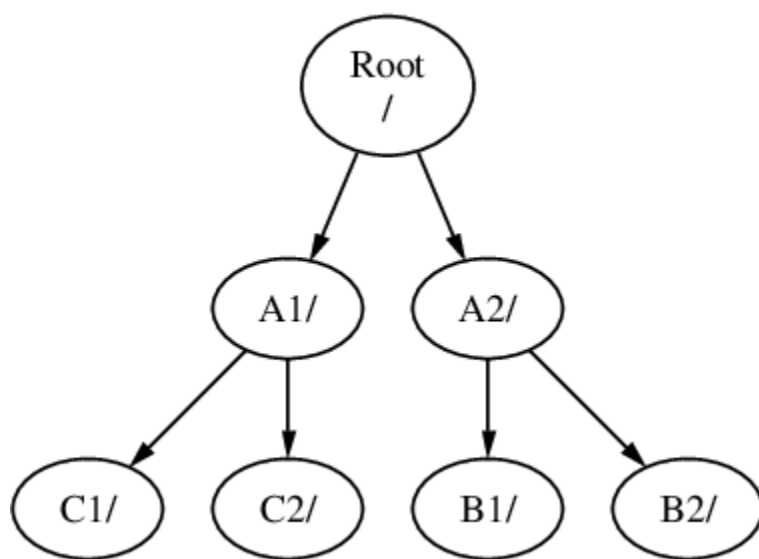
and the paths would be `/A2/B1` and `/A2/B2` respectively.

File systems can be mounted on top of one another. Continuing the last example, the C file system could be mounted on top of the B1 directory in the B file system, leading to this arrangement:





Or C could be mounted directly on to the A file system, under the A1 directory:



It is entirely possible to have one large root file system, and not need to create any others. There are some drawbacks to this approach, and one advantage.

Benefits of Multiple File Systems

- Different file systems can have different *mount options*. For example, the root file system can be mounted read-only, making it impossible for users to inadvertently delete or edit a critical file. Separating user-writable file systems, such as `/home`,

from other file systems allows them to be mounted *nosuid*. This option prevents the *suid/guid* bits on executables stored on the file system from taking effect, possibly improving security.

- FreeBSD automatically optimizes the layout of files on a file system, depending on how the file system is being used. So a file system that contains many small files that are written frequently will have a different optimization to one that contains fewer, larger files. By having one big file system this optimization breaks down.
- FreeBSD's file systems are robust if power is lost. However, a power loss at a critical point could still damage the structure of the file system. By splitting data over multiple file systems it is more likely that the system will still come up, making it easier to restore from backup as necessary.


Benefit of a Single File System

- File systems are a fixed size. If you create a file system when you install FreeBSD and give it a specific size, you may later discover that you need to make the partition bigger. This is not easily accomplished without backing up, recreating the file system with the new size, and then restoring the backed up data.

Important

FreeBSD features the [growfs\(8\)](#) command, which makes it possible to increase the size of file system on the fly, removing this limitation. A file system can only be expanded into free space in the partition in which it resides. If there is space after the partition, the partition can be expanded with [gpart\(8\)](#). If the partition is the last one on a virtual disk, and the disk is expanded, the partition can then be expanded.

3.6.2. Disk partitions

File systems are contained in *partitions*. Disks are divided into partitions using one of several partitioning schemes; see [Manual Partitioning](#). The newer scheme is GPT; older BIOS-based computers use MBR. GPT supports division of a disk into partitions with a size, offset, and type. It supports a large number of partitions and partition types, and is recommended whenever its use is possible. GPT partitions use the disk name with a suffix, where the suffix is *p1* for the first partition, *p2* for the second, and so on. MBR  however, supports only a small number of partitions. The MBR partitions are known in

FreeBSD as `slices`. Slices may be used for different operating systems. FreeBSD slices are subdivided into partitions using BSD labels (see [bsdlabel\(8\)](#)).

Slice numbers follow the device name, prefixed with an `s`, starting at 1. So `"da0s1"` is the first slice on the first SCSI drive. There can only be four physical slices on a disk, but there can be logical slices inside physical slices of the appropriate type. These extended slices are numbered starting at 5, so `"ada0s5"` is the first extended slice on the first SATA disk. These devices are used by file systems that expect to occupy a slice.

Each GPT or BSD partition can contain only one file system, which means that file systems are often described by either their typical mount point in the file system hierarchy, or the name of the partition they are contained in.

FreeBSD also uses disk space for *swap space* to provide *virtual memory*. This allows your computer to behave as though it has much more memory than it actually does. When FreeBSD runs out of memory, it moves some of the data that is not currently being used to the swap space, and moves it back in (moving something else out) when it needs it. This is called *paging*.

Some BSD partitions have certain conventions associated with them.

Partition	Convention
a	Normally contains the root file system.
b	Normally contains swap space.
c	Normally the same size as the enclosing slice. This allows utilities that need to work on the entire slice, such as a bad block scanner, to work on the <code>c</code> partition. A file system would not normally be created on this partition.



d

Partition **d** used to have a special meaning associated with it, although that is now gone and **d** may work as any normal partition.

Slices and "dangerously dedicated" physical drives contain BSD partitions, which are represented as letters from **a** to **h**. This letter is appended to the device name, so "da0a" is the **a** partition on the first **da** drive, which is "dangerously dedicated". "ada1s3e" is the fifth partition in the third slice of the second SATA disk drive.

Finally, each disk on the system is identified. A disk name starts with a code that indicates the type of disk, and then a number, indicating which disk it is. Unlike partitions and slices, disk numbering starts at 0. Common codes are listed in [Disk Device Names](#).

When referring to a partition in a slice, include the disk name, **s**, the slice number, and then the partition letter. Examples are shown in [Sample Disk](#). GPT partitions include the disk name, **p**, and then the partition number.

[Conceptual Model of a Disk](#) shows a conceptual model of a disk layout using MBR slices.

When installing FreeBSD, configure the disk slices if using MBR, and create partitions within the slice to be used for FreeBSD. If using GPT, configure partitions for each file system. In either case, create a file system or swap space in each partition, and decide where each file system will be mounted. See [gpart\(8\)](#) for information on manipulating partitions.

Table 3. Disk Device Names

Drive Type	Drive Device Name
SATA and IDE hard drives	ada
SCSI hard drives and USB storage devices	da
NVMe storage	nvd or nda



Drive Type	Drive Device Name
SATA and IDE CD-ROM drives	cd
SCSI CD-ROM drives	cd
Floppy drives	fd
SCSI tape drives	sa
RAID drives	Examples include aacd for Adaptec® AdvancedRAID, mlx and mlyd for Mylex®, amrd for AMI MegaRAID®, idad for Compaq Smart RAID, twed for 3ware® RAID.

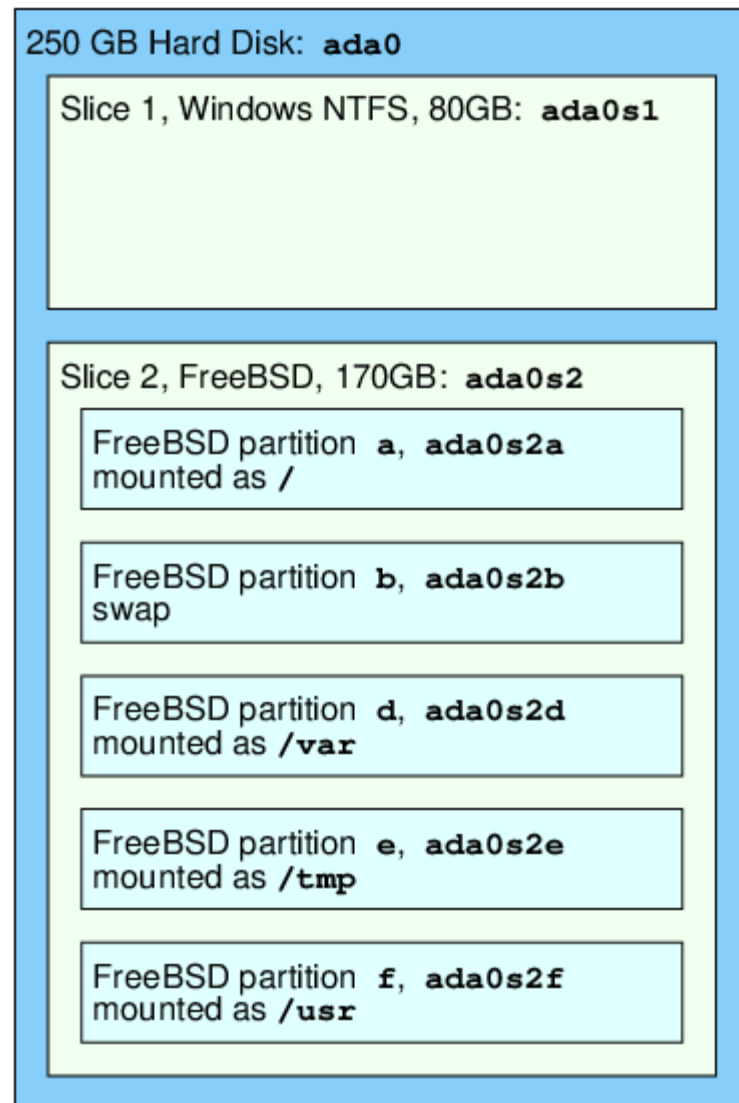
Table 4. Sample Disk, Slice, and Partition Names

Name	Meaning
ada0s1a	The first partition (a) on the first slice (s1) on the first SATA disk (ada0).
da1s2e	The fifth partition (e) on the second slice (s2) on the second SCSI disk (da1).

Example 12. Conceptual Model of a Disk

This diagram shows FreeBSD's view of the first SATA disk attached to the system. Assume that the disk is 250 GB in size, and contains an 80 GB slice and a 170 GB slice (MS-DOS® partitions). The first slice contains a Windows® NTFS file system, C: , and the second slice contains a FreeBSD installation. This example FreeBSD installation has four data partitions and a swap partition.

The four partitions each hold a file system. Partition **a** is used for the root file system, **d** for `/var/`, **e** for `/tmp/`, and **f** for `/usr/`. Partition letter **c** refers to the entire slice, and so is not used for ordinary partitions.



3.7. Mounting and Unmounting File Systems

The file system is best visualized as a tree, rooted, as it were, at `/`. `/dev`, `/usr`, and the other directories in the root directory are branches, which may have their own branches, such as `/usr/local`, and so on.



There are various reasons to house some of these directories on separate file systems. `/var` contains the directories `log/`, `spool/`, and various types of temporary files, and as such, may get filled up. Filling up the root file system is not a good idea, so splitting `/var` from `/` is often favorable.

Another common reason to contain certain directory trees on other file systems is if they are to be housed on separate physical disks, or are separate virtual disks, such as Network File System mounts, described in “[Network File System \(NFS\)](#)”, or CDROM drives.

3.7.1. The `fstab` File

During the boot process ([The FreeBSD Booting Process](#)), file systems listed in `/etc/fstab` are automatically mounted except for the entries containing `noauto`. This file contains entries in the following format:

device	/mount-point	fstype	options	dumpfreq	passn
--------	--------------	--------	---------	----------	-------

device

An existing device name as explained in [Disk Device Names](#).

mount-point

An existing directory on which to mount the file system.

fstype

The file system type to pass to [mount\(8\)](#). The default FreeBSD file system is `ufs`.

options

Either `rw` for read-write file systems, or `ro` for read-only file systems, followed by any other options that may be needed. A common option is `noauto` for file systems not normally mounted during the boot sequence. Other options are listed in [mount\(8\)](#).

dumpfreq

Used by [dump\(8\)](#) to determine which file systems require dumping. If the field is missing, a value of zero is assumed.



passno

Determines the order in which UFS file systems should be checked by [fsck\(8\)](#) after a reboot. File systems that should be skipped should have their `passno` set to zero. The root file system needs to be checked before everything else and should have its `passno` set to one. The other file systems should be set to values greater than one. If more than one file system has the same `passno`, [fsck\(8\)](#) will attempt to check file systems in parallel if possible.

Refer to [fstab\(5\)](#) for more information on the format of `/etc/fstab` and its options.

3.7.2. Using [mount\(8\)](#)

File systems are mounted using [mount\(8\)](#). The most basic syntax is as follows:

```
# mount device mountpoint
```



A file system listed in `/etc/fstab` can also be mounted by providing just the mountpoint.

This command provides many options which are described in [mount\(8\)](#). The most commonly used options include:

Mount Options

-a

Mount all the file systems listed in `/etc/fstab`, except those marked as "noauto", excluded by the `-t` flag, or those that are already mounted.

-d

Do everything except for the actual mount system call. This option is useful in conjunction with the `-v` flag to determine what [mount\(8\)](#) is actually trying to do.

-f

Force the mount of an unclean file system (dangerous), or the revocation of write access when downgrading a file system's mount status from read-write to read-



only.

-r

Mount the file system read-only. This is identical to using `-o ro`.

-t *fstype*

Mount the specified file system type or mount only file systems of the given type, if `-a` is included. "ufs" is the default file system type.

-u

Update mount options on the file system.

-v

Be verbose.

-w

Mount the file system read-write.

The following options can be passed to `-o` as a comma-separated list:

nosuid

Do not interpret setuid or setgid flags on the file system. This is also a useful security option.

3.7.3. Using **umount(8)**

To unmount a file system use **umount(8)**. This command takes one parameter which can be a mountpoint, device name, `-a` or `-A`.

All forms take `-f` to force unmounting, and `-v` for verbosity. Be warned that `-f` is not generally a good idea as it might crash the computer or damage data on the file system.

To unmount all mounted file systems, or just the file system types listed after `-t`, use `-a` or `-A`. Note that `-A` does not attempt to unmount the root file system.

3.8. Processes and Daemons



FreeBSD is a multi-tasking operating system. Each program running at any one time is

called a *process*. Every running command starts at least one new process and there are a number of system processes that are run by FreeBSD.

Each process is uniquely identified by a number called a *process ID* (PID). Similar to files, each process has one owner and group, and the owner and group permissions are used to determine which files and devices the process can open. Most processes also have a parent process that started them. For example, the shell is a process, and any command started in the shell is a process which has the shell as its parent process. The exception is a special process called `init(8)` which is always the first process to start at boot time and which always has a PID of `1`.

Some programs are not designed to be run with continuous user input and disconnect from the terminal at the first opportunity. For example, a web server responds to web requests, rather than user input. Mail servers are another example of this type of application. These types of programs are known as *daemons*. The term daemon comes from Greek mythology and represents an entity that is neither good nor evil, and which invisibly performs useful tasks. This is why the BSD mascot is the cheerful-looking daemon with sneakers and a pitchfork.

There is a convention to name programs that normally run as daemons with a trailing "d". For example, BIND is the Berkeley Internet Name Domain, but the actual program that executes is named `.` The Apache web server program is `httpd` and the line printer spooling daemon is `lpd`. This is only a naming convention. For example, the main mail daemon for the Sendmail application is `sendmail`, and not `maild`.

3.8.1. Viewing Processes

To see the processes running on the system, use `ps(1)` or `top(1)`. To display a static list of the currently running processes, their PIDs, how much memory they are using, and the command they were started with, use `ps(1)`. To display all the running processes and update the display every few seconds in order to interactively see what the computer is doing, use `top(1)`.

By default, `ps(1)` only shows the commands that are running and owned by the user. For example:

```
% ps
```



The output should be similar to the following:

```

PID TT  STAT    TIME COMMAND
8203  0   Ss    0:00.59 /bin/csh
8895  0   R+    0:00.00 ps

```

The output from `ps(1)` is organized into a number of columns. The `PID` column displays the process ID. PIDs are assigned starting at 1, go up to 99999, then wrap around back to the beginning. However, a PID is not reassigned if it is already in use. The `TT` column shows the tty the program is running on and `STAT` shows the program's state. `TIME` is the amount of time the program has been running on the CPU. This is usually not the elapsed time since the program was started, as most programs spend a lot of time waiting for things to happen before they need to spend time on the CPU. Finally, `COMMAND` is the command that was used to start the program.

A number of different options are available to change the information that is displayed. One of the most useful sets is `auxww`, where `a` displays information about all the running processes of all users, `u` displays the username and memory usage of the process' owner, `x` displays information about daemon processes, and `ww` causes `ps(1)` to display the full command line for each process, rather than truncating it once it gets too long to fit on the screen.

The output from `top(1)` is similar:

```
% top
```



The output should be similar to the following:

```

last pid: 9609; load averages: 0.56, 0.45, 0.36                up
107 processes: 2 running, 104 sleeping, 1 zombie
CPU: 6.2% user, 0.1% nice, 8.2% system, 0.4% interrupt, 85.1% id
Mem: 541M Active, 450M Inact, 1333M Wired, 4064K Cache, 1498M Free
ARC: 992M Total, 377M MFU, 589M MRU, 250K Anon, 5280K Header, 21M 0t
Swap: 2048M Total, 2048M Free

```

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	C	TIME	WCPU
557	root	1	-21	r31	136M	42296K	select	0	2:20	9.96%
8198	dru	2	52	0	449M	82736K	select	3	0:08	5.96%
8311	dru	27	30	0	1150M	187M	uwait	1	1:37	0.98%
431	root	1	20	0	14268K	1728K	select	0	0:06	0.93%
9551	dru	1	21	0	16600K	2660K	CPU3	3	0:01	0.98%

```

2357  dru          4  37      0   718M   141M select    0    0:21  0.00%
8705  dru          4  35      0   480M    98M select    2    0:20  0.00%
8076  dru          6  20      0   552M   113M uwait     0    0:12  0.00%
2623  root         1  30     10 12088K   1636K select    3    0:09  0.00%
2338  dru          1  20      0   440M  84532K select    1    0:06  0.00%
1427  dru          5  22      0   605M  86412K select    1    0:05  0.00%

```

The output is split into two sections. The header (the first five or six lines) shows the PID of the last process to run, the system load averages (which are a measure of how busy the system is), the system uptime (time since the last reboot) and the current time. The other figures in the header relate to how many processes are running, how much memory and swap space has been used, and how much time the system is spending in different CPU states. If the ZFS file system module has been loaded, an `ARC` line indicates how much data was read from the memory cache instead of from disk.

Below the header is a series of columns containing similar information to the output from `ps(1)`, such as the PID, username, amount of CPU time, and the command that started the process. By default, `top(1)` also displays the amount of memory space taken by the process. This is split into two columns: one for total size and one for resident size. Total size is how much memory the application has needed and the resident size is how much it is actually using now.

`top(1)` automatically updates the display every two seconds. A different interval can be specified with `-s`.

3.8.2. Killing Processes

One way to communicate with any running process or daemon is to send a *signal* using `kill(1)`. There are a number of different signals; some have a specific meaning while others are described in the application's documentation. A user can only send a signal to a process they own and sending a signal to someone else's process will result in a permission denied error. The exception is the `root` user, who can send signals to anyone's processes.

The operating system can also send a signal to a process. If an application is badly written and tries to access memory that it is not supposed to, FreeBSD will send the process the "Segmentation Violation" signal (`SIGSEGV`). If an application has been written to use the `alarm(3)` system call to be alerted after a period of time has elapsed, it will be sent the "Alarm" signal (`SIGALRM`).



Two signals can be used to stop a process: `SIGTERM` and `SIGKILL`. `SIGTERM` is the polite way to kill a process as the process can read the signal, close any log files it may have open, and attempt to finish what it is doing before shutting down. In some cases, a process may ignore `SIGTERM` if it is in the middle of some task that cannot be interrupted.

`SIGKILL` cannot be ignored by a process. Sending a `SIGKILL` to a process will usually stop that process there and then. ^[1].

Other commonly used signals are `SIGHUP`, `SIGUSR1`, and `SIGUSR2`. Since these are general purpose signals, different applications will respond differently.

For example, after changing a web server's configuration file, the web server needs to be told to re-read its configuration. Restarting `httpd` would result in a brief outage period on the web server. Instead, send the daemon the `SIGHUP` signal. Be aware that different daemons will have different behavior, so refer to the documentation for the daemon to determine if `SIGHUP` will achieve the desired results.

! Important

Killing a random process on the system is a bad idea. In particular, `init(8)`, PID 1, is special. Running `/bin/kill -s KILL 1` is a quick, and unrecommended, way to shutdown the system. *Always* double check the arguments to `kill(1)` before pressing .

3.9. Shells

A *shell* provides a command line interface for interacting with the operating system. A shell receives commands from the input channel and executes them. Many shells provide built in functions to help with everyday tasks such as file management, file globbing, command line editing, command macros, and environment variables. FreeBSD comes with several shells, including the Bourne shell (`sh(1)`) and the extended C shell (`tcsh(1)`). Other shells are available from the FreeBSD Ports Collection, such as `zsh` and `bash`.

The shell that is used is really a matter of taste. A C programmer might feel more comfortable with a C-like shell such as `tcsh(1)`. A Linux® user might prefer `bash`. Each shell has unique properties that may or may not work with a user's preferred working environment, which is why there is a choice of which shell to use.



One common shell feature is filename completion. After a user types the first few letters of a command or filename and presses `Tab`, the shell completes the rest of the command or filename. Consider two files called `foobar` and `football`. To delete `foobar`, the user might type `rm foo` and press `Tab` to complete the filename.

But the shell only shows `rm foo`. It was unable to complete the filename because both `foobar` and `football` start with `foo`. Some shells sound a beep or show all the choices if more than one name matches. The user must then type more characters to identify the desired filename. Typing a `t` and pressing `Tab` again is enough to let the shell determine which filename is desired and fill in the rest.

Another feature of the shell is the use of environment variables. Environment variables are a variable/key pair stored in the shell's environment. This environment can be read by any program invoked by the shell, and thus contains a lot of program configuration.

[Common Environment Variables](#) provides a list of common environment variables and their meanings. Note that the names of environment variables are always in uppercase.

Table 5. Common Environment Variables

Variable	Description
USER	Current logged in user's name.
PATH	Colon-separated list of directories to search for binaries.
DISPLAY	Network name of the Xorg display to connect to, if available.
SHELL	The current shell.
TERM	The name of the user's type of terminal. Used to determine the capabilities of the terminal.



TERMCAP	Database entry of the terminal escape codes to perform various terminal functions.
OSTYPE	Type of operating system.
MACHTYPE	The system's CPU architecture.
EDITOR	The user's preferred text editor.
PAGER	The user's preferred utility for viewing text one page at a time.
MANPATH	Colon-separated list of directories to search for manual pages.

How to set an environment variable differs between shells. In [tcsh\(1\)](#) and [csh\(1\)](#), use `setenv` to set environment variables. In [sh\(1\)](#) and `bash`, use `export` to set the current environment variables. This example sets the default `EDITOR` to `/usr/local/bin/emacs` for the [tcsh\(1\)](#) shell:

```
% setenv EDITOR /usr/local/bin/emacs
```



The equivalent command for `bash` would be:

```
% export EDITOR="/usr/local/bin/emacs"
```



To expand an environment variable in order to see its current setting, type a `$` character in front of its name on the command line. For example, `echo $TERM` displays the current `$TERM` setting.

Shells treat special characters, known as meta-characters, as special representations of



data. The most common meta-character is `*`, which represents any number of characters in a filename. Meta-characters can be used to perform filename globbing. For example, `echo *` is equivalent to `ls` because the shell takes all the files that match `*` and `echo` lists them on the command line.

To prevent the shell from interpreting a special character, escape it from the shell by starting it with a backslash (`\`). For example, `echo $TERM` prints the terminal setting whereas `echo \$TERM` literally prints the string `$TERM`.

3.9.1. Changing the Shell

The easiest way to permanently change the default shell is to use `chsh`. Running this command will open the editor that is configured in the `EDITOR` environment variable, which by default is set to `vi(1)`. Change the `Shell:` line to the full path of the new shell.

Alternately, use `chsh -s` which will set the specified shell without opening an editor. For example, to change the shell to `bash`:

```
% chsh -s /usr/local/bin/bash
```



Enter your password at the prompt and press to change your shell. Log off and log in again to start using the new shell.

Note

The new shell *must* be present in `/etc/shells`. If the shell was installed from the FreeBSD Ports Collection as described in [Installing Applications: Packages and Ports](#), it should be automatically added to this file. If it is missing, add it using this command, replacing the path with the path of the shell:

```
# echo /usr/local/bin/bash >> /etc/shells
```



Then, rerun `chsh(1)`.



3.9.2. Advanced Shell Techniques

The UNIX® shell is not just a command interpreter, it acts as a powerful tool which allows users to execute commands, redirect their output, redirect their input and chain commands together to improve the final command output. When this functionality is mixed with built in commands, the user is provided with an environment that can maximize efficiency.

Shell redirection is the action of sending the output or the input of a command into another command or into a file. To capture the output of the [ls\(1\)](#) command, for example, into a file, redirect the output:

```
% ls > directory_listing.txt
```



The directory contents will now be listed in `directory_listing.txt`. Some commands can be used to read input, such as [sort\(1\)](#). To sort this listing, redirect the input:

```
% sort < directory_listing.txt
```



The input will be sorted and placed on the screen. To redirect that input into another file, one could redirect the output of [sort\(1\)](#) by mixing the direction:

```
% sort < directory_listing.txt > sorted.txt
```



In all of the previous examples, the commands are performing redirection using file descriptors. Every UNIX® system has file descriptors, which include standard input (stdin), standard output (stdout), and standard error (stderr). Each one has a purpose, where input could be a keyboard or a mouse, something that provides input. Output could be a screen or paper in a printer. And error would be anything that is used for diagnostic or error messages. All three are considered I/O based file descriptors and sometimes considered streams.

Through the use of these descriptors, the shell allows output and input to be passed around through various commands and redirected to or from a file. Another method of redirection is the pipe operator.



The UNIX® pipe operator, "`|`", allows the output of one command to be directly passed or directed to another program. Basically, a pipe allows the standard output of a command to be passed as standard input to another command, for example:



```
% cat directory_listing.txt | sort | less
```

In that example, the contents of `directory_listing.txt` will be sorted and the output passed to [less\(1\)](#). This allows the user to scroll through the output at their own pace and prevent it from scrolling off the screen.

3.10. Text Editors

Most FreeBSD configuration is done by editing text files, so it is a good idea to become familiar with a text editor. FreeBSD comes with a few as part of the base system, and many more are available in the Ports Collection.

A simple editor to learn is [ee\(1\)](#), which stands for easy editor. To start this editor, type `ee filename` where *filename* is the name of the file to be edited. Once inside the editor, all of the commands for manipulating the editor's functions are listed at the top of the display. The caret (`^`) represents `Ctrl`, so `^e` expands to `Ctrl + e`. To leave [ee\(1\)](#), press `Esc`, then choose the "leave editor" option from the main menu. The editor will prompt to save any changes if the file has been modified.

FreeBSD also comes with more powerful text editors, such as [vi\(1\)](#), as part of the base system. Other editors, like [editors/emacs](#) and [editors/vim](#), are part of the FreeBSD Ports Collection. These editors offer more functionality at the expense of being more complicated to learn. Learning a more powerful editor such as vim or Emacs can save more time in the long run.

Many applications which modify files or require typed input will automatically open a text editor. To change the default editor, set the `EDITOR` environment variable as described in [Shells](#).

3.11. Devices and Device Nodes

A device is a term used mostly for hardware-related activities in a system, including disks, printers, graphics cards, and keyboards. When FreeBSD boots, the majority of the boot messages refer to devices being detected. A copy of the boot messages is saved to `/var/run/dmesg.boot`.



Each device has a device name and number. For example, `ada0` is the first SATA hard drive, while `kbd0` represents the keyboard.

Most devices in FreeBSD must be accessed through special files called device nodes, which are located in `/dev`.

3.12. Manual Pages

The most comprehensive documentation on FreeBSD is in the form of manual pages. Nearly every program on the system comes with a short reference manual explaining the basic operation and available arguments. These manuals can be viewed using `man`:

```
% man command
```



where *command* is the name of the command to learn about. For example, to learn more about `ls(1)`, type:

```
% man ls
```



Manual pages are divided into sections which represent the type of topic. In FreeBSD, the following sections are available:

1. User commands.
2. System calls and error numbers.
3. Functions in the C libraries.
4. Device drivers.
5. File formats.
6. Games and other diversions.
7. Miscellaneous information.
8. System maintenance and operation commands.
9. System kernel interfaces.



In some cases, the same topic may appear in more than one section of the online

manual. For example, there is a `chmod` user command and a `chmod()` system call. To tell [man\(1\)](#) which section to display, specify the section number:

```
% man 1 chmod
```



This will display the manual page for the user command [chmod\(1\)](#). References to a particular section of the online manual are traditionally placed in parenthesis in written documentation, so [chmod\(1\)](#) refers to the user command and [chmod\(2\)](#) refers to the system call.

If the name of the manual page is unknown, use `man -k` to search for keywords in the manual page descriptions:

```
% man -k mail
```



This command displays a list of commands that have the keyword "mail" in their descriptions. This is equivalent to using [apropos\(1\)](#).

To read the descriptions for all of the commands in `/usr/sbin`, type:

```
% cd /usr/sbin
% man -f * | more
```



or

```
% cd /usr/sbin
% whatis * |more
```



3.12.1. GNU Info Files

FreeBSD includes several applications and utilities produced by the Free Software Foundation (FSF). In addition to manual pages, these programs may include hypertext documents called `info` files. These can be viewed using [info\(1\)](#) or, if [editors/emacs](#) is installed, the `info` mode of `emacs`.



To use [info\(1\)](#), type:

```
% info
```



For a brief introduction, type `h`. For a quick command reference, type `?`.

1. There are a few tasks that cannot be interrupted. For example, if the process is trying to read from a file that is on another computer on the network, and the other computer is unavailable, the process is said to be uninterruptible. Eventually the process will time out, typically after two minutes. As soon as this time out occurs the process will be killed.

Last modified on: February 13, 2025 by [Pouria Mousavizadeh Tehrani](#)

[< Prev](#)

[🏠 Home](#)

[Next >](#)



English

System



About

[FreeBSD](#)

[FreeBSD Foundation](#)

[Get FreeBSD](#)

[Code of Conduct](#)

[Security Advisories](#)



Documentation

[Documentation portal](#)

[Manual pages](#)

[Presentations and papers](#)

[Previous versions](#)

[4.4BSD Documents](#)

[Wiki](#)

Community

[Get involved](#)

[Community forum](#)

[Mailing lists](#)

[IRC Channels](#)

[Bug Tracker](#)

Legal

[Donations](#)

[Licensing](#)

[Privacy Policy](#)

[Legal notices](#)

© 1994-2025 The FreeBSD Project. All rights reserved

Made with  by the FreeBSD Community

