Summary of common Unix commands

John W. Shipman

2008-01-02 19:11

Abstract

Describes commands commonly used in Unix-type operating systems.

This publication is available in Web form[1] and also as a PDF document[2]. Please forward any comments to **`tcc-doc@nmt.edu`**.

Table of Contents

1. How to use this document

An online tutorial[3] accompanies this list of commands. These web pages will introduce you to the basic concepts of Unix commands. The rest of this document will serve as a reference.

2. Typographical conventions

Actual command input is shown in **`typewriter type`**.

Arguments are shown in *`italics`*.

Items that are optional are shown in [square brackets].

The ellipsis symbol, "…", means that the preceding item can be repeated.

[1] http://www.nmt.edu/tcc/help/pubs/unixcrib/
[2] http://www.nmt.edu/tcc/help/pubs/unixcrib/unixcrib.pdf
[3] http://www.nmt.edu/tcc/help/unix/fund.html

3. Parts of a command

After you log in, you will get a *prompt*, that is, a request to type a command. The part of Unix that reads and executes your command is called a *shell*, and there are several different shells. Most users use the *bash* shell, which prompts with a dollar sign "$". Some people prefer *tcsh*, which prompts with a percent sign "%". Type this command to find out which shell you have:

```
echo $0
```

Here are the parts of a Unix command:

```
commandname [argument]… [<inputfile] [> outputfile]
```

commandname
: The name of the command comes first.

argument
: If there are any arguments to the command, they follow the command name.

inputfile
: If there is an input redirection symbol "<", input comes from ***inputfile***, otherwise it comes from the keyboard.

outputfile
: If there is an output redirection symbol ">", output is sent to ***outputfile***, otherwise it is sent to the screen.

It is also possible to link up several commands so that the output of each command is used as the input of the next command:

```
command1 | command2 | … | commandn
```

In this example, the output of ***command***$_1$ is used as the input for ***command***$_2$; the output of ***command***$_2$ is used as the input for ***command***$_3$; and so on.

4. Organizing your files

4.1. Commands for the current working directory

`ls -F [`*`file`*`…]`
: List your files (in columns)

`ls -l [`*`file`*`…]`
: List your files (in detail).

`ls -al`
: Also list all the hidden files (those whose names start with a period).

`cp` ***`oldname newcopy`***
: Copy a file.

`mv` ***`oldname newname`***
: Rename or move a file.

4.2. Working with directories

At any given time, you will be located in some *current working directory*. When you first log in, this will be your account's *home directory*. You can create subdirectories under that, so you can keep files for different projects separate. You can create subdirectories within subdirectories, and so on to any depth.

You will also need to know how to use directories so that you can use files located under other users' accounts.

A *pathname* is a description of a file's location somewhere in the Unix directory structure.

- Most file and path names are assumed to be in the current working directory. Slashes (“`/`”) are used to denote subdirectories, so for example this pathname

  ```
  bill/cat
  ```

  refers to file `cat` in directory `bill` under the current working directory.

- Pathnames that start with a tilde character “~” are relative to your home directory. For example,

  ```
  ~/foo
  ```

  refers to file `foo` in your home directory.

- Pathnames that start with “~”a tilde followed by another user's name are relative to that user's home directory (assuming that you have permission to see into that directory). For example,

  ```
  ~moriarty/bar/klarn
  ```

  refers to file `klarn` in directory `bar` under user `moriarty`'s home directory.

- You can refer to the parent directory by using two dots (“..”), so pathname

  ```
  ../ack
  ```

  refers to file `ack` in the directory above the current working directory. Similarly, the pathname

  ```
  ../../letters/susie
  ```

  refers to file `susie` in subdirectory `letters` under the directory two levels above the current working directory.

- Pathnames starting with a slash (“`/`”`/`) are called *absolute pathnames*. Your home directory will typically be located at absolute path `/u/`*`yourname`*, where ***`yourname`*** is your account (login) name. (The path reported by the **`pwd`** command may be different, but that's an artifact of our local software.)

Commands for working with directories:

`cd` ***`pathname`***
  Change the current working directory.

`pwd`
  Show the pathname of the current working directory.

`mv` ***`file`****...*
  Move the given files to a different directory.

`mkdir` ***`path`***
  Create a new directory.

`rmdir` ***`path`***
  Remove an empty directory.

`ln -s oldfile newname`
Make ***`newname`*** a symbolic link to ***`oldfile`***.

4.3. Controlling access to your account

The first line of defense against others tampering with your files is proper user authentication. Don't tell anyone your password, and don't write it down. To change your password, use this command:

```
passwd
```

Also, *always log out* when you are finished. From an X window session, move the mouse to the background, press and hold the right button, and select *Logout*.

4.4. Controlling access to your files

You can control whether others can see or change your files. This is done by setting *permissions* on a file-by-file basis. There are three kinds of permissions:

- If someone has *read* permission for a file, that means they can see its contents.
- *Write* permission is the power to change or delete a file.
- *Execute* permission applies to programs and commands—the ability to execute a file as a program or command. (Execute permission for a directory grants the right to see what files are there.)

To find out the permissions for a file, use this command:

```
ls -l [file]…
```

In the output from this command, the first ten characters show you the file type and permissions.

- The first character is “`-`” for an ordinary file, “`d`” for a directory, or “`l`” for a symbolic link.
- The next three characters specify the read (“`r`”), write (“w”), and execute (“`x`”) permissions for the file's owner. A hyphen (“`-`”) means no permission.
- The next three characters specify the read, write, and execute permissions for group members. Groups are a mechanism for allowing file access to a specific list of people; if you need to set up this kind of access, e-mail a request to **`tcc-eng@nmt.edu`**.
- The next three characters specify the read, write, and execute permissions for the “world” (all other users).

The command to change permissions is:

```
chmod who=value file…
```

where:

`who`
is ***`u`*** to set the owner's permissions, ***`g`*** to set group permissions, ***`o`*** to set world (“other”) permissions, and ***`a`*** to set all three at once.

`value`
can be up to three of the letters ***`r`*** for read permissions, ***`w`*** for write permissions, and ***`x`*** for execute permissions.

file…
  is a list of files whose permissions are to be changed.

To remove permissions, use:

```
chmod who-value file…
```

And to add permissions:

```
chmod who+value file…
```

5. Controlling multiple processes

Normally, a command is executed in the *foreground*, which means it can read input from the keyboard and send messages to the screen.

If you have a process that does not need to read input from the keyboard, you can run it in the *background*, without tying up your window. To do this, end the command with an ampersand ("**&**"):

```
command&
```

The command will be assigned a *job number* that will be displayed in square brackets. You can use this job number to control the process.

Commands for process control:

control-Z
  Suspend the current foreground job.

stop *jobno*
  Suspend job ***jobno***, but it can be resumed.

kill %*jobno*
  Terminate ***jobno***.

bg %*jobno*
  Resume ***jobno*** in the background.

fg %*jobno*
  Resume ***jobno*** in the foreground.

jobs
  Show all the current background jobs.

ps -gx
  Show all your processes on this machine by *process ID* number.

kill -9 *pid*
  Kill a process using its process ID number.