Darren Provine at Rowan University (Lab Tech)

File Management Matters

Keeping track of your files, and keeping them organized, becomes increasingly important as you work on larger and larger projects. It's not really very hard under Unix to manage your files: they can be moved, renamed, copied, scanned, examined, and protected in a variety of ways with only a few commands. All that's necessary is to remember the relevant commands.

Ways to look at files

cat - output the contents of a file

-v
verbose - show control characters with ^ notation

-e
mark the **e**nd of the line with a $

-t
show **t**ab characters as ^I

more - display output in screen-sized chunks

There are many options, but more(1) is generally run as is; the defaults are pretty reasonable. The most important commands to use when running more are:

SPACE	down one page
b	up one page
d	down one-half page
RETURN	down one line
q	quit

head - show the beginning of output or a file

Given a numeric flag, such as **-15**, head shows that many lines. With no flag, it defaults to 10.

tail - show the end of output or a file

Given a numeric flag, such as **-15**, tail shows that many lines. With no flag, it defaults to 10.

The **-f** flag causes tail to **f**ollow the file's progress as it is updated. To stop, hit ^C.

file - identify a file's contents
This program opens a file and attempts to figure out what's in it.

wc - word count
Display the number of lines, words, and chars in a file.

Ways to move/copy/delete files

cp - copy files or directories
The most common flags used with cp(1) are:
-i
cause the program to **i**nquire before stomping another file
-R
cause the program to **r**ecurse and copy an entire directory subtree

mv - move files or directories
The flag usually used with mv is **-i**, which (as with cp) causes the program to **i**nquire before destroying something else. There is no flag for moving or renaming an entire directory, since mv does that by default.

rm - remove files or directories
This program has three option flags you need to know about:
-i
Cause the program to **I**nquire before removing a file.
-r
Cause the program to **R**ecurse and remove an entire directory subtree. Use with caution!
-f
Cause the program to **F**orce a removal without asking, even if the file is read-only. Use only with extreme caution!

touch - update modification time of a file
This program sets the modification time of a file to `right now'. If the file doesn't exist, an empty file will be created.

mkdir - create a directory
With the **-p** flag, you can create the **p**ath to a directory. That is, you can say "mkdir -p foo/bar/baz", and it will make both foo and foo/bar, if needed.

rmdir - remove a directory

This only removes a directory if it is empty. Much safer than using rm with the -r option.

Ways to look at your directory arrangement

ls - list files and directories

This program has a huge array of options. Most common:

-l
long format listing
-d
information about the **d**irectory, not the files in it
-r
reverse the order of the output
-t
sort files by **t**ime instead of name
-a
show **a**ll files, even those whose names begin with a period (usually these are elided)
-F
marks **f**ile types with special chars; *e.g.*, a directory has a / appended to its name
-s
include file **s**ize in blocks with output

pwd - print working directory

Print the absolute pathname of your current location in the directory tree.

tree - draw a directory tree

Tree will produce a graphical representation of the named directory, or . if no directory is named. Common flags include:

-a
show **A**ll files
-p
show **P**ermissions
-L #
only go # **L**evels deep
-I pattern
Ignore files that match *pattern* (probably need pattern in quotes)

find - scan the directory tree for files

"find" is unusual among the programs we're looking at, because the other programs require that option flags appear *before* the names of files that they work on; in "find", the flags that control behaviour come *after* the list of directories to be searched. So you would type "find [directory list] [predicate list]".

"find" will examine every file in the directories named, and will test those files against the predicate list. If each of the tests is found to be true, then the file's name is printed.

If no directories are named, "find" will search the current directory. If no predicates are listed, "find" will print the name of every file.

The most useful predicates include:

-atime [+-]*number*
True if the file was **a**ccessed **number** days ago. A plus means "more than **number**", and a minus means "less than **number**".
-mtime [+-]*number*
As above, but this tests when the contents of the file were **m**odified.
-ctime [+-]*number*
As above, but this tests when the file's inode was **c**hanged. For example, if you change the permissions on a file, that affects ctime but not mtime.
-name '*name*'
True if the file is named **name**. Shell filename matches (such as *, ?, and []) can be used. There is another version, **-iname**, which does case-insensitive searches.
-user '*loginid* '
True if the file is owned by the user **loginid**.
-size [+-]'*number*[c]'
True if the file is of size **number** disk blocks. As with the time comparisons, a plus means that the file is bigger than **number**, and a minus means that the file is smaller than **number**. A "c" at the end of the number means that you want the number of **C**haracters, not disk blocks.
-exec '*command;*'
If other tests are found true, this will execute **command**. To specify the name of the file as part of the command, you have to use "{ }". Do not omit the semicolon at the end of the command.

Predicates can be grouped with parentheses; an exclamation point ("!") means "not". Two predicates together are combined with logical AND; to get logical OR, use "-o" in between them.

locate - find files by name

This program works by scanning a list of names, which is updated every week or so. It can't find files by time or permission, but it's extremely fast if you know the name of the file you need.

Note that the master list will not include files in protected directories, so if you are looking for something in such a directory, you'll need find.

Setting permissions

chmod - change a file's mode

Change the mode bits (which specify permissions) of the named file or directory according to the specified permission setting.

There are two methods of setting permissions: octal and mnemonic.

The permissions on a file are specified by nine bits, in three sets of three. Consider this permission string (copied from the output of ls -l): `rw-r--r--` This means that the **user** has read and write permission, people in the file's **group** have read permissions, and **others** have read permissions. In binary (base 2), that's 110100100; in octal (base 8), it's 644.

To set these permissions for a file, you could run `chmod 644` *`filename`*.

A problem with octal permissions is that they set *all* of the bits at once. You might want to turn some bits on or off, without also changing any others. In that case, you can use the mnemonic method, by specifying which class of user you want to change, and what change you want to make.

For example, if a file has permissions `rwxr-xr-x`, and you want to turn off read and execute permission for everyone but yourself, you could run `chmod go-rx` *`filename`*. That would turn off read and execute permission for **g**roup and **o**thers, without changing your permissions.