Linux Filesystem Quick-Reference

Directory Contents with ls

All of these take one or more directories (relative or absolute) as an argument. If not supplied, the current directory is used. Many installations have a default shell alias for ls to supply some common options. A nice default is 'ls -FCA'.

Command	*Result*
ls	List the directory contents
ls -l	Long listing, with lots of details
ls -a	Include files/directories beginning with a dot
ls -A	Like -a, but omitting special . and .. directories
ls -lt	Long listing, sorted by modified time (latest first)
ls -ltr	Like -lt, but reversing the sort order
ls -1	Produce a listing in a single column
ls -m	Produce a listing as a single comma-delimited line
ls -F	Decorate names with type indicators (/=directory, @=link, etc.)
ls -lh	Display sizes in friendly units
ls -ln	Display owner and group as numbers, not names
ls -d	Do not expand directories

Moving Around

Command	*Result*
cd	Change the working directory to the user's home directory
cd *dir*	Change the working directory to *dir*
cd ~*user*	Change the working directory to *user*'s home directory
cd -	Change the working directory to the previous working directory
pushd *dir*	Like "cd *dir*", but adds *dir* to bash's directory stack
popd	Pop the top of the directory stack, cd'ing to the new top

Moving, Copying, and Extracting Pieces of Files

Command	*Result*
cp *a* *b*	Create a copy of file *a* named *b*
mv *a* *b*	Rename file *a* so that it is now *b*
dd if=*a* of=*b*	Dump the contents of input file *a* to output file *b*

For all of these the files may be in any directory, so to move a file `foo` up a directory, you could run

```
mv foo ../foo
```

or just

```
mv foo ../
```

`dd` is an extremely powerful tool, and `if` and `of` are only the beginning of its options. Either can be omitted, and use STDIN or STDOUT as the default. Here's another simple example:

```
dd if=/dev/zero of=foo bs=1024 count=5
```

This will create a file `foo` with the first 5kB of `/dev/zero`, which will provide you with as many NULL bytes as you request. Give this a try, and then run

```
xxd foo
```

See the manpage for `dd` for all options. It's well worth you time to learn more about this command!

File Permissions

Every file or directory has an user (owner) and group, and a set of permission bits (the first column of "ls -l"). On most systems, your group will be the same as your username, though other groups are likely to exist, and you may be a member of some of them. The `groups` command will show you what groups your account belongs to.

Here are some examples from the Fall 2018 course VM:

vmuser@f18marsh:~$ ls -ld gitsrc drwxrwxr-x 21 vmuser vmuser 4096 Jun 21 15:16 gitsrc vmuser@f18marsh:~$ ls -ld .ssh drwx—— 2 vmuser vmuser 4096 Jun 21 14:18 .ssh vmuser@f18marsh:~$ ls -l .ssh total 12 -rw——- 1 vmuser vmuser 3312 Jun 21 14:17 id_rsa -rw-rw-r– 1 vmuser vmuser 738 Jun 21 14:18 id_rsa.pub -rw-r–r– 1 vmuser vmuser 444 Jun 21 14:18 known_hosts

In all of these, vmuser is the owner, and all files/directories are also assigned to the group vmuser. The first column is 10-characters wide:

Character	*Meaning*
0	File type: d=directory, l=symlink, c=char device
1	User (u) read (r) permission
2	User (u) write (w) permission
3	User (u) execute (x) permission
4	Group (g) read (r) permission
5	Group (g) write (w) permission
6	Group (g) execute (x) permission
7	Other (o) read (r) permission
8	Other (o) write (w) permission
9	Other (o) execute (x) permission

Anything not set is indicated with a "-", which for character 0 means a normal file. We see that gitsrc is a directory, readable and executable by everyone (user, group, and other), but writable only by user and group. For directories, "executable" means a user with matching credentials can cd into that directory. ~/.ssh/id_rsa, a private key, has full permissions for the user, but no permissions for anyone else. ~/.ssh/id_rsa.pub is readable by everyone, and also writable by the group.

We can change the permissions on a file (a directory is just a type of file) using `chmod`. Here are some options:

Option	*Meaning*
u+rwx	Add read, write, and execute perms for the user
g+rwx	The same, for the group
o+rwx	The same, for others
o-w	Removed write permissions for others
go-rwx	Remove all permissions for the group and others
ugo+x	Add execute perms for all users
a+x	The same as the previous
700	Set the permissions to -rwx——

Option	*Meaning*
655	Set the permissions to -rwxr-xr-x
-R	Apply the permissions recursively, when given a directory

The numeric versions set permissions exactly, and use octal to specify the bits (1=x, 2=w, 4=r) in the order (user, group, other). After writing a lot of scripts, `chmod a+x <file>` will become part of your muscle memory.

You can also change the ownership of files, using `chown`. The syntax is

```
chown <user>:<group> <file>
```

When you run things as root, you often have to run this (using `sudo`) to fix the file ownership. As with `chmod`, you can provide -R to change ownership recursively.

Disk Usage

These will let you figure out how much space is used/available, and where that used space is.

Command	*Result*
df	Display statistics for all mounted filesystems
df *dir*	Display statistics for the filesystem on which *dir* is mounted
df -h	Use friendly units for sizes
du *dir*	Count the disk usage for the specified directory and subdirs
du -s	Only show the total usage, not the subdir breakdown
du -h	Use friendly units for sizes

Special Files

Most executables live in /bin, /usr/bin, or /usr/local/bin

Most libraries (static or shared) live in /lib, /usr/lib, or /usr/local/lib

Configuration files generally live in /etc

Temporary files generally live in /tmp, which is often flushed on shutdown

Device files live in /dev, and a couple of these are worth note:

- /dev/null contains nothing, and is often used as a target for output that should be discarded
- /dev/zero will produce as many null bytes as you care to read

Process files live in /proc, in subdirectories named with process IDs (PIDs). Also of possible interest in /proc (somewhat-readable ASCII files):

- /proc/cpuinfo
- /proc/meminfo
- /proc/stat
- /proc/vmstat

Finding Things

Being able to find something specific is extremely useful. Here are some tools to do this:

- `locate <name>` – Given *name*, find indexed files containing *name* as a substring; relies on `updatedb` having been run since the file was added.
- `find <dir> ...` – Starting in *dir*, find files matching a set of specifiers. More on this below.
- `grep <pattern> <files>` – Find lines in *files* matching *pattern*. More on this below.
- `ack <pattern> [<dir>]` – Like grep, but faster when searching large directories. Most systems don't have this installed by default.

`grep` can take regular expressions, and can operate recursively on directories, though it tends not to be particularly efficient when doing so. Here are some options (there are many more):

Option	*Meaning*
-E	Interpret *pattern* as an extended regular expression
-r	Recursively grep directories
-A *n*	Include *n* lines of context after a matching line
-B *n*	Include *n* lines of context before a matching line
-C *n*	Include *n* lines of context before and after a matching line
-H	Prepend matching lines with the name of the file
-i	Ignore case in matches
-l	Only print the names of files with matches
-L	Only print the names of files without matches
-n	Prepend the matching line number
-q	Don't print matches, just return 0 (match) or -1 (no match)
-v	Match lines **not** including *pattern*

`find` has a *lot* of options, far too many to go into detail here. Some of the more useful ones:

Option	*Meaning*
-name *n*	Match files containing *n*
-iname *n*	Case-insensitive version of -name
-type *t*	Match files of type *t* (f=normal file, d=directory, etc.)
-depth *d*	Limit the depth of the search
-size *s*	Match files with size matching *s*, like 10, 20k, 32M, etc.
-size -*s*	Match files smaller than *s*
-size +*s*	Match files larger than *s*
-newer *f*	Match files modified more recently than file *f*
-mtime *t*	Match files modified within time *t*, default unit days
	Also, -*t* or +*t*
	-ctime and -atime do same thing for file creation and access
-print	Print the name of a matched file (default)
-ls	Print `ls -l`-like lines for matching files
-exec ...	Execute a command on matches (see below)
-delete	Removes files and directories - **USE WITH EXTREME CAUTION**

For matches, the order can matter, especially for performance. You want to run -exec as late in the filtering process as possible, for example, since it runs an external program for each file.

-exec is very powerful, because it allows you to extend find's already-considerably functionality. Here's an illustrative example:

```
find . -name \*.txt -exec grep -H foo {} \;
```

This will start from the current directory, match all files ending in ".txt", and run grep on them. The string "{}" is replaced with the name of the current match. The -exec command must be terminated with ";", regardless of whether any other commands are provided. This is essentially the same as:

```
grep --include \*.txt -Hr foo .
```