

# 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'`.

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

## Moving Around

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

## 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:

| <i>Character</i> | <i>Meaning</i>                                   |
|------------------|--|
| 0                | File type: d=directory, l=symlink, c=char device |

| <i>Character</i> | <i>Meaning</i>                   |
|------------------|----------------------------------|
| 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:

| <i>Option</i>       | <i>Meaning</i>  |
|---------------------|---|
| <code>u+rwX</code>  | Add read, write, and execute perms for the user           |
| <code>g+rwX</code>  | The same, for the group                                   |
| <code>o+rwX</code>  | The same, for others                                      |
| <code>o-w</code>    | Removed write permissions for others                      |
| <code>go-rwx</code> | Remove all permissions for the group and others           |
| <code>ugo+x</code>  | Add execute perms for all users                           |
| <code>a+x</code>    | The same as the previous                                  |
| <code>700</code>    | Set the permissions to <code>-rwx---</code>               |
| <code>655</code>    | Set the permissions to <code>-rwxr-xr-x</code>            |
| <code>-R</code>     | 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.

| <i>Command</i>              | <i>Result</i>  |
|-----------------------------|--|
| <code>df</code>             | Display statistics for all mounted filesystems                                     |
| <code>df &lt;dir&gt;</code> | Display statistics for the filesystem on which <code>&lt;dir&gt;</code> is mounted |
| <code>df -h</code>          | Use friendly units for sizes   |
| <code>du &lt;dir&gt;</code> | Count the disk usage for the specified directory and subdirs                       |

| <i>Command</i>     | <i>Result</i>                                       |
|--------------------|---|
| <code>du -s</code> | Only show the total usage, not the subdir breakdown |
| <code>du -h</code> | 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):

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

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

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

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 .
```