Level Up!
Unix Terminal & Filesystem

Host: Drew Dunne

Table of contents

Notation

- Commands will be shown on slides using `teletype text`.

Introducing new commands

`some-command [opt1] [opt2]`

New commands will be introduced in block boxes like this one, sometimes including common flags or warnings.

- To execute `some-command`, just type its name into the shell and press return / enter.
- `$` in code-blocks indicate a new command being entered.

```
$ first-command
output of first-command (where applicable)
$ second-command
output of second-command (where applicable)
```

Unix Filesystem Overview

The Unix Filesystem

- Unlike Windows, UNIX has a single global "root" directory (instead of a root directory for each disk or volume).
    - The root directory is just /
- All files and directories are case sensitive.
    - hello.txt != hElLo.TxT
- Directories are separated by / instead of \ in Unix.
    - UNIX: /home/sven/lemurs
    - Windows: E:\Documents\lemurs
- Hidden files and folders begin with a "."
    - e.g. .git/ (a hidden directory).
- Example: my home directory.

What's Where: Programs Edition

Programs are usually installed in one of the "binaries" directories:

- **/bin**: System programs.
- **/usr/bin**: Most user programs.
- **/usr/local/bin**: A few other user programs.

- Your personal files are in your home directory (and its subdirectories), which is *usually** located at

Linux	Mac
/home/username	/Users/username

- There is also a built-in alias for it: ~
- For example, the Desktop for the user sven is located at

Linux	Mac
/home/sven/Desktop	/Users/sven/Desktop
~/Desktop	~/Desktop

Basic Navigational Commands

- Most shells default to using the current path in their prompt. If not, you can find out where you are with

Print working directory

pwd

- Prints the "full" path of the current directory.
- Handy on minimalist systems when you get lost.
- Can be used in scripts.

- Note that if you have a path with *symbolic* links, you need to use the -P flag.

What's here?

- Knowing where you are is useful, but understanding what else is there is too...

The list command

`ls`

- Lists directory contents (including subdirectories).

- Works like the dir command in Windows.

- The `-l` flag lists detailed file / directory information (we'll learn more about flags later).

- Use `-a` to list hidden files.

Ok lets go!

- Moving around is as easy as

Changing directories

`cd [directory name]`

- Changes directory to `[directory name]`.

- If not given a destination defaults to the user's home directory.

- You can specify both absolute and relative paths.

- If you do not specify a `directory`, the `~` (home) `directory` is assumed.

- Absolute paths start at `/` (the global root).
  - e.g. `cd /home/sven/Desktop`
- Relative paths start at the current directory.
  - e.g. `cd Desktop`, if you were already at `/home/sven`

Relative Path Shortcuts

- Shortcuts

~	current user's home directory
.	the current directory (this is actually useful…)
..	the parent directory of the current directory
-	for **cd** command, return to previous working directory

- An example: starting in **/usr/local/src**

```
$ cd     # now at /home/sven
$ cd -   # now at /usr/local/src
$ cd ..  # now at /usr/local
```

File and Folder Manipulation

Creating a new File

- The easiest way to create an empty file is using

touch

```
touch [flags] <file>
```

- Adjusts the timestamp of the specified file.

- With no flags uses the current date and time.

- If the file does not exist, **touch** creates it.

- File extensions (.txt, .c, .py, etc) often **don't** matter in Unix. Using **touch** to create a file results in a blank plain-text file (so you don't necessarily have to add .txt to it).

- No magic here...

Make directory

```
mkdir [flags] <dir1> <dir2> <...> <dirN>
```

- Can use relative or absolute paths.

  - a.k.a. you are not restricted to making directories in the current directory only.

- Need to specify at least one directory name.

- Can specify multiple, separated by spaces.

- The -p flag is commonly used in scripts:

  - Makes all parent directories if they do not exist.
  - Convenient because if the directory exists, mkdir will not fail.

File Deletion

- Warning: once you delete a file (from the command line) there is no easy way to recover the file.

Remove File

`rm [flags] <filename>`

- Removes the file `<filename>`.

- Remove multiple files with wildcards (more on this later).

  - Remove every file in the current directory: `rm *`
  - Remove every `.jpg` file in the current directory: `rm *.jpg`

- Prompt before deletion: `rm -i <filename>`

Deleting Directories

- By default, **rm** cannot remove directories. Instead we use...

Remove directory

```
rmdir [flags] <directory>
```

- Removes an **empty** directory.

- Throws an error if the directory is not empty.

- You are encouraged to use this command: failing on non-empty can and will save you!

- To delete a directory and all its subdirectories, we pass **rm** the flag **-r** (for recursive), e.g. `rm -r /home/sven/oldstuff`

Copy

`cp [flags] <file> <destination>`

- Copies from one location to another.
- To copy multiple files, use wildcards (such as *).
- To copy a complete directory: `cp -r <src> <dest>`

- Unlike the **cp** command, the move command automatically recurses for directories.
  - Think of the implication of if it did not...

Move

```
mv [flags] <source> <destination>
```

- Moves a file or directory from one place to another.

- Also used for renaming, just move from **<oldname>** to **<newname>**.

  - E.g. mv badFolderName correctName

`ls`	list directory contents
`cd`	change directory
`pwd`	print working directory
`rm`	remove file
`rmdir`	remove directory
`cp`	copy file
`mv`	move file

Flags & Command Clarifaction

Flags and Options

- Most commands take flags and optional arguments.
- These come in two general forms:
    - Switches (no argument required), and
    - Argument specifiers (for lack of a better name).
- When specifying flags for a given command, keep in mind:
    - Flags modify the behavior of the command / how it executes.
    - Some flags take precedence over others, and some flags you specify can implicitly pass additional flags to the command.

A flag that is

- One letter is specified with a single dash (-a).
- More than one letter is specified with two dashes (--all).
- The reason is because of how switches can be combined (next page).

Flags and Options: Switches

Switches take no arguments, and can be specified in a couple of different ways. Switches are usually one letter, and multiple letter switches usually have a one letter alias (the `ls` command has `--all` aliased to `-a`).

- One option:
    - `ls -a`
    - `ls --all`
- Two options:
    - `ls -l -Q`
- Two options:
    - `ls -lQ`
- *Usually* applied from left to right in terms of operator precedence, but not always:
    - This is up to the developer of the tool.
    - `rm -fi <file>` ⇒ prompts
    - `rm -if <file>` ⇒ does *not* prompt

Flags and Options: Argument Specifiers

- These flags expect an input, and you will encounter two general kinds.
- The `--argument="value"` format, where the `=` and quotes are needed if **value** is more than one word.
    - Yes: `ls --hide="Desktop" ~/`
    - Yes: `ls --hide=Desktop ~/`
        - one word, no quotes necessary
    - No: `ls --hide = "Desktop" ~/`
        - spaces by the `=` will be misinterpreted (it used `=` as the `hide` value...)
- The `--argument value` format, with a space after the `argument`. Quote rules same as above.
    - `ls --hide "Desktop" ~/`
    - `ls --hide Desktop ~/`

* Note: The example I gave you was using the same `--hide` in both formats, but not *all* commands will accept both.

Advise `--argument="value"` format for higher success rates.

Generally, you should always specify the flags before the arguments. In this example, the flag is `-l` and `~/Desktop/` is the argument.

- `ls -l ~/Desktop/` and `ls ~/Desktop/ -l` both work
- there exist scenarios in which flags after arguments do **not** get processed

There is a special sequence `--` that signals the end of the options. I will use another flag to demonstrate:

- `ls -l -a ~/Desktop/` ⇒ executes as expected
- `ls -l -- -a ~/Desktop/` ⇒ only used `-l`
  - "ls: cannot access -a: No such file or directory"
  - `-a` was treated as an *argument*, and there is no `-a` directory (for me)

The special sequence `--` that signals the end of the options is often most useful if you need to do something special. Suppose I wanted to make the folder `-a` on my Desktop.

```
$ cd ~/Desktop # for demonstration purpose
$ mkdir -a     # fails: invalid option -- 'a'
$ mkdir -- -a  # success! (ls to confirm)
$ rmdir -a     # fails: invalid option -- 'a'
$ rmdir -- -a  # success! (ls to confirm)
```

This trick can be useful in *many* scenarios, and generally arises when you need to work with special characters of some sort.

Your new best friend

How do I know what the flags / options for all of these commands are?

The manual command

man <command_name>

- Loads the manual (manpage) for the specified command.
- Unlike google, manpages are **system-specific**.
- Usually very comprehensive. Sometimes *too* comprehensive.
- Type **/<keyword>** to search.
- The **n** key jumps through the search results.

Search example on next page if that was confusing. Intended for side-by-side follow-along.

26

Users and Groups

Like most OS's, Unix allows multiple people to use the same machine at once. The question: who has access to what?

- Access to files depends on the users' account.
- All accounts are presided over by the Superuser, or root account.
- Each user has absolute control over any files they own, which can only be superseded by root.
- Files can also be owned by a group, allowing more users to have access.

File Ownership

- You can discern who owns a file many ways, the most immediate being `ls -l`

Permissions with `ls`

```
$ ls -l Makefile
-rw-rw-r--. 1 sven users 4.9K Jan 31 04:42 Makefile
              sven        # the user
                    users # the group
```

- The third column is the *user*, and the fourth column is the *group.*

What is this RWX Nonsense?

- R = read, W = write, X = execute.
- rwxrwxrwx
    - User permissions.
    - Group permissions.
    - Other permissions (a.k.a. neither the owner, nor a member of the group).
- Directory permissions begin with a **d** instead of a -.

What would the permissions `-rwxr-----` mean?

- It is a file.
- User can read and write to the file, as well as execute it.
- Group members are allowed to read the file, but cannot write to or execute.
- Other cannot do *anything* with it.

Change Mode

`chmod <mode> <file>`

- Changes file / directory permissions to **<mode>**.
- The format of **<mode>** is a combination of three fields:
  - Who is affected: a combination of **u**, **g**, **o**, or **a** (all).
  - Use a **+** to add permissions, and a **-** to remove.
  - Specify type of permission: any combination of **r**, **w**, **x**.
- Or you can specify mode in octal: user, then group, then other.
  - e.g. **777** means user=7, group=7, other=7 permissions.

The octal version can be confusing, but will save you time. Excellent resource in [2].

Changing the group

Change Group

```
chgrp group <file>
```

- Changes the group ownership of `<file>` to `group`.

As the super user, you can change who owns a file:

Change Ownership

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

- Changes the ownership of `<file>`.
- The `group` is optional.
- The `-R` flag is useful for recursively modifying everything in a directory.

[1] B. Abrahao, H. Abu-Libdeh, N. Savva, D. Slater, and others over the years.
Previous cornell cs 2043 course slides.