02 - The Unix File System

CS 2043: Unix Tools and Scripting, Spring 2017 [1]

Stephen McDowell

January 27th, 2017

Cornell University

Table of contents

1. Unix Filesystem Overview

2. Basic Navigational Commands

3. File and Folder Manipulation

4. Flags & Command Clarifaction

Some Logistics

- Virtual Machines are *almost* ready, need to update the videos.

Some Logistics

- Virtual Machines are *almost* ready, need to update the videos.
  - Assuming minimal errors recording, afternoon - dinner time.

Some Logistics

- Virtual Machines are *almost* ready, need to update the videos.
    - Assuming minimal errors recording, afternoon - dinner time.
- Still on the fence about taking the class?

Some Logistics

- Virtual Machines are *almost* ready, need to update the videos.
  - Assuming minimal errors recording, afternoon - dinner time.
- Still on the fence about taking the class?
  - Moral obligation: decide now, many others want to enroll.

Notation

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

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.

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.

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.

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 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 **/**

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.

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`

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.

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`

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`

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 ”`.`”

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).

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?

- `/dev`: Hardware devices, like your hard drive, USB devices.

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices.
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices.
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives.

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices.
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives.
- `/usr`: Mostly user-installed programs and amenities.

What's Where?

- `/dev`: Hardware devices, like your hard drive, USB devices.
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives.
- `/usr`: Mostly user-installed programs and amenities.
- `/etc`: System-wide settings.

What's Where: Programs Edition

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

- `/bin`: System programs.

What's Where: Programs Edition

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

- `/bin`: System programs.
- `/usr/bin`: Most user programs.

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.

Personal Files

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

Linux	Mac
`/home/username`	`/Users/username`

Personal Files

- 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: `~`

Personal Files

- 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

Where am I?

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

Where am I?

- 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
```

Where am I?

- 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.

Where am I?

- 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.

Where am I?

- 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.

Where am I?

- 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...

What's here?

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

The list command

```
ls
```

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).

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.

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).

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

Ok lets go!

- Moving around is as easy as

Changing directories

```
cd [directory name]
```

Ok lets go!

- Moving around is as easy as

Changing directories

```
cd [directory name]
```

- Changes directory to `[directory name]`.

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.

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.

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.

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.

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).

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`

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.

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

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`

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

Creating a new File

- The easiest way to create an empty file is using

touch

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

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.

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.

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.

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).

Creating a new Directory

- No magic here...

Creating a new Directory

- No magic here...

Make directory

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

Creating a new Directory

- No magic here...

Make directory

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

- Can use relative or absolute paths.

Creating a new Directory

- 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.

Creating a new Directory

- 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.

Creating a new Directory

- 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.

Creating a new Directory

- 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:

Creating a new Directory

- 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.

Creating a new Directory

- 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.

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>
```

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>`.

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).

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 *`

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`

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...

Deleting Directories

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

Remove directory

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

Deleting Directories

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

Remove directory

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

- Removes an **empty** directory.

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.

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!

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 That!

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>`

Move it!

- Unlike the **cp** command, the move command automatically recurses for directories.

Move it!

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

Move it!

- 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>
```

Move it!

- 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.

Move it!

- 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>`.

Move it!

- 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`

Recap

`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.

Flags and Options

- Most commands take flags and optional arguments.
- These come in two general forms:

Flags and Options

- Most commands take flags and optional arguments.
- These come in two general forms:
    - Switches (no argument required), and

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).

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 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.

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.

Flags and Options: A bad Analogy

- If you think of a command as a computer, you could think of the flags as the different hardware components installed. Let's say that in this case a hard drive is a flag.

Flags and Options: A bad Analogy

- If you think of a command as a computer, you could think of the flags as the different hardware components installed. Let's say that in this case a hard drive is a flag.
- The computer shipped to you with a CPU, motherboard, hard drive, etc and installed on that hard drive was the original operating system (say Windows). When you start it, the computer was executed with the Windows flag.

Flags and Options: A bad Analogy

- If you think of a command as a computer, you could think of the flags as the different hardware components installed. Let's say that in this case a hard drive is a flag.
- The computer shipped to you with a CPU, motherboard, hard drive, etc and installed on that hard drive was the original operating system (say Windows). When you start it, the computer was executed with the Windows flag.
- Now, you remove the original hard drive and insert another hard drive that has a different OS installed (say Fedora). Then you boot your computer, only this time you ended up passing the Fedora flag.

Flags and Options: A bad Analogy

- If you think of a command as a computer, you could think of the flags as the different hardware components installed. Let's say that in this case a hard drive is a flag.
- The computer shipped to you with a CPU, motherboard, hard drive, etc and installed on that hard drive was the original operating system (say Windows). When you start it, the computer was executed with the Windows flag.
- Now, you remove the original hard drive and insert another hard drive that has a different OS installed (say Fedora). Then you boot your computer, only this time you ended up passing the Fedora flag.
- Nothing about the other components of the computer changed (it's just a bunch of electricity being routed around), but the behavior changed because of the flag you passed.

Flags and Options: Formats

A flag that is

- One letter is specified with a single dash (`-a`).

Flags and Options: Formats

A flag that is

- One letter is specified with a single dash (`-a`).
- More than one letter is specified with two dashes (`--all`).

Flags and Options: Formats

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`).

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:

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`

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`

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:

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`

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:

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`

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:

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.

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>` $\Rightarrow$ prompts

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.

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.

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" ~/`

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 ~/`

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

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" ~/`

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...)

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.

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" ~/`

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 ~/`

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.

Flags and Options: Conventions, Warnings

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

Flags and Options: Conventions, Warnings

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

Flags and Options: Conventions, Warnings

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

Flags and Options: Conventions, Warnings

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:

Flags and Options: Conventions, Warnings

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/` $\Rightarrow$ executes as expected

Flags and Options: Conventions, Warnings

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`

Flags and Options: Conventions, Warnings

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"`

Flags and Options: Conventions, Warnings

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)

Flags and Options: Conventions, Warnings (cont)

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.

Flags and Options: Conventions, Warnings (cont)

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)
```

Flags and Options: Conventions, Warnings (cont)

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.

Man oh man

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
############### [first result highlighted]
$ n       #  followed by enter
############### [next result highlighted]
```

Man oh man

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
############## [first result highlighted]
$ n       #  followed by enter
############## [next result highlighted]
```

Note that there are subtle differences between options on different systems. For example, `ls -B`:

Man oh man

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
############## [first result highlighted]
$ n       #  followed by enter
############## [next result highlighted]
```

Note that there are subtle differences between options on different systems. For example, `ls -B`:

- BSD/OSX: Force printing of non-printable characters in file names as **\xxx**, where **xxx** is the numeric value of the character in *octal*.

Man oh man

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
############## [first result highlighted]
$ n       #  followed by enter
############## [next result highlighted]
```

Note that there are subtle differences between options on different systems. For example, `ls -B`:

- BSD/OSX: Force printing of non-printable characters in file names as `\xxx`, where `xxx` is the numeric value of the character in *octal*.
- Fedora, Ubuntu: do not list implied entries ending with `~`

Man oh man

```
$ man man #  you now have the manual loaded
$ /useful #  type /useful, then hit enter
############## [first result highlighted]
$ n       #  followed by enter
############## [next result highlighted]
```

Note that there are subtle differences between options on different systems. For example, `ls -B`:

- BSD/OSX: Force printing of non-printable characters in file names as `\xxx`, where `xxx` is the numeric value of the character in *octal*.
- Fedora, Ubuntu: do not list implied entries ending with `~`
  - In these OS's, files ending with `~` are *temporary* backup files that certain programs (e.g. some text-editors) generate.

References I

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