

02 – Navigating the Unix File System

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

January 25, 2019

Cornell University

Table of Contents

1. So you've logged in. Or are sitting next to someone who has.
2. Our first commands: navigating the filesystem
3. Where to go: The Unix Filesystem
4. Let's use some files (and directories!)

So you've logged in. Or are sitting
next to someone who has.

Your place in the file system: where am I?

What you should see now (modulo colors)

```
NetID@wash ~ $
```

- *NetID* is your username
- **wash** is the *hostname* of the computer you're accessing
- **~** is the path to your current *directory*
 - (we call folders "directories" in *nix land because AT&T invented these words)

- This is the **bash prompt**, the default command line.
- everything in bash is based on a *current directory*
- You are currently *inside* the **~ Directory**. What does this mean?
 - **~** is a special symbol for your *home* directory
 - you own everything in your home directory
 - (on personal computers) contains Desktop, Downloads, etc.

What's in a command?

- Commands work like functions for bash
- Command is a single word, like **command**
- Commands can take arguments
 - arguments are space-separated:
 - **command arg1 arg2** passes **arg1** and **arg2** to **command**
- **Most arguments are optional**
- *position-independent* arguments are called “flags” and are prefixed with a - or --
- example: **command --flag**
- example: **command -f**

Notation

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

Introducing New Commands

some-command **[opt1]** **[opt2]** **<arg1>** **[arg2]**

- New commands will be introduced in block boxes like this one
 - **[brackets]** indicate *optional* items (flags / arguments)
 - **<arg1>**: **arg1** is required
 - **[arg2]**: command supports multiple arguments
- To execute **some-command**, just type its name into the shell and press return / enter.
 - **\$** in code-blocks indicate a new command being entered.

\$ some-command

output of some-command (where applicable)

Our first commands: navigating the filesystem

Where am I?

- Most shells (including ours) 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.
 - The **-P** flag is needed when *symbolic* links are present.
- Handy on minimalist systems when you get lost.
- Can be used in scripts.

What's here?

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

List Directory Contents

`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 let's go!

- Moving around is as easy as

Change Directories

```
cd [directory name]
```

- Changes directory to **[directory name]**.
- If not given a destination defaults to the user's home directory.
 - Reminder: the home directory is ~

A bit on paths

- A *path* describes how to access a file
- Most paths are *relative* paths – they start in your current working directory
- Simple paths are just file names in the current directory
 - example: I'm in ~, which contains **course**; while I'm in ~ the path **course** will refer to this directory
- A path can *traverse* directories using the / separator
 - example: the path ~/**course** will *always* mean the directory **course** in my home directory, no matter what my current working directory is.
 - example: to get to the directory **bar** in the directory **baz** in the directory ~, I could `cd ~/bar/baz`.

Relative Path Shortcuts

- Relative path shortcuts worth remembering:

Shortcut	Expands To
~	current user's home directory
.	the current directory
..	the parent directory of the current directory
-	for cd , return to previous working directory

- An example:
 - ~/course/cs2043 arbitrary choice, nothing special about it.
 - After each **cd** command, execute **pwd** to confirm.

```
$ cd ~/course/cs2043 # go to starting location
$ cd                 # now at /home/mpm288
$ cd -               # now at ~/course/cs2043
$ cd ..              # now at ~/course
```

Where to go: The Unix Filesystem

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 / in Unix instead of \ in Windows.
 - UNIX: `/home/mpm288/lemurs`
 - Windows: `E:\Documents\lemurs`
- Absolute paths start with a /, and always refer to the root directory (and never care about the current working directory)
- Hidden files and directories begin with a “.”
 - e.g. `.git/` (a hidden directory)
 - e.g. `..` (your parent directory)

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 (access) disk drives.
 - Your second hard drive, for example. Instead of E:\, **/mnt/better_name_than_E**
- **/media**: For accessing removable storage drives, like flash drives, CDs, etc.
 - instead of D:\, **/media/optical_drive**
- **/usr**: Mostly user-installed programs and amenities.
- **/etc**: System-wide settings.

What's Where: Programs Edition

- Programs *usually* installed in one of the “binaries” directories:
 - `/bin`: System programs.
 - `/usr/bin`: System-managed user programs.
 - `/usr/local/bin`: Manually-installed user programs

Personal Files

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

Linux	Mac
<code>/home/username</code>	<code>/Users/username</code>

- There is also a built-in alias for it: `~`
- For example, the course for the user **mpm288** is located at

Linux	Mac
<code>/home/mpm288/course</code>	<code>/Users/mpm288/course</code>
<code>~/course</code>	<code>~/course</code>

Let's use some files (and directories!)

Printing a file

- What good is moving around with reading stuff?

Concatenate files and print them

```
cat [files]...
```

- Prints (“concatenates”) the listed files to your terminal
 - With no arguments, does something more advanced
- **note:** if you run **cat** without any arguments and your console is just hanging, hold **CTRL** and press **C** to stop the program.
 - This works in general to stop programs.
 - try to **cat** the file **README** in your home directory!
 - **READMEs** are generally important files. Read them if you want information!

Creating a new File

- The easiest way to create an empty file is using

Change File Timestamps

```
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.
 - “But I swear I haven’t changed the file, look at the timestamp.”
 - ... timestamps prove nothing.
- 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.
 - You don’t have to add **.txt** if you don’t want to.

Creating a new Directory

- No magic here...

Make Directories

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

- Can use relative or absolute paths.
 - 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 Files or Directories

`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

```
rm -r [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)
 - **rm -r /home/mpm288/oldstuff**
 - THIS IS DANGEROUS!

Copy That!

Copy

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

- Copies from one location to another.
- To copy multiple files, use wildcards (such as *).
 - Globbs / patterns can only be used for **<src>**.
 - **<dest>** must be explicit and singularly defined.
 - Completely reasonable...how would it know what to do if there is ambiguity in where to send the file(s)?
- To copy a complete directory: **cp -r <src> <dest>**

Move it!

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

Move (or Rename) Files and Directories

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

- Moves a file or directory from one place to another.
- Also used for renaming, rename **<oldname>** to **<newname>**.
 - `mv badFolderName correctName`

Hand it in!

- For CS2043, we've written a special command **handin** to turn in your assignments

hand in your homework

```
handin <assignment> <file_name>
```

- Hands in a *single file* for the named assignment
- You can never hand in more than one file per assignment

check if you **handed in** your homework

```
check-handin <assignment>
```

Recap

<code>ls</code>	list directory contents
<code>cd</code>	change directory
<code>pwd</code>	print working directory
<code>rm</code>	remove file
<code>rmdir</code>	remove directory
<code>cp</code>	copy file
<code>mv</code>	move file
<code>handin</code>	hand in homework
<code>check-handin</code>	check if handin worked

References

- [1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. “Previous Cornell CS 2043 Course Slides”.