Command Line Basics

This lesson is meant to provide students with an introduction to the command line.

Programmers of all kinds live on the command line. It gives us fast and reliable control over computers. the CLI (command line interface) has become a sort of lingua franca of computer programming. Because so many developers spend so much time in the CLI there are an abundance of commands that programmers have developed to make programming easier.

Not only that, but Web servers usually don't have graphical interfaces, so we are forced to interact with them through command line and programmatic interfaces. Once you become comfortable using the command line, staying on the keyboard will also help you keep an uninterrupted flow of work going without the disruption of shifting to the mouse.

Objectives -- You Should Learn How To:

- Describe the connection between the command line and the "Graphical User Interface"
- Use `man`, "*the only command you'll ever need*."
- Use www.explainshell.com.
- Perform basic file manipulation & navigation operations on the command line:
  - `pwd, ls, cd, touch, mkdir, rm, rmdir, cp, mv`
- Search a file with `grep`.
- Chain commands and redirect output: `|, >, >>`.
- Search your bash history with `history | grep` and `ctrl+r`.
- Describe permissions and change them.
  - Use `sudo, chmod, chown`
- Explore interesting, but non-essential commands on your own:
  - networks: `ping, curl, wget, traceroute`
  - systems: `ps, top, df`

Topics

- Introduction
  - What is the Terminal
  - Opening the Terminal Application
- Current Working Directory
  - Current Directory
  - Home Directory
  - `ls`, `pwd`, commands
- Navigating Around
  - Root Directory
  - `cd`
  - Absolute and Relative Paths
  - Tab Completion
- File Manipulation
  - `mkdir`
  - Editing Files
  - echo, Redirection and Piping

Introduction

What is the Terminal?

Terminal is a modern version of an 'original' `User Interface` for unix based computers. At that time a Text Terminal is all you would have seen, no windows, no mouse. Because of this history, it's very powerful but sometimes a little cryptic.

Don't worry though, with a bit of practice you'll be flying around like a pro!

Although they technically mean slightly different things, the following terms are synonyous with the Terminal Environment:

- Shell
- bash ('Bourne-Again shell', although I've haven't heard that used recently)
- Command Line
- Text Terminal
- DOS Prompt (on windows machines)
- SSH (on remote machines)
- Bourne Shell
- csh
- ksh
- sh
- UNIX Shell

Opening the Terminal

Follow Along:

1. In the top right of the screen click the Magnifing Glass icon (or use `⌘+space`) to bring up 'Spotlight' and type 'Terminal'
2. Once Terminal starts locate the icon in the doc and select `Options->Keep In Dock` so that it's always handy.

Note: In documentation we often see a `#` or a `$` prefix before code examples, these characters are used to indicate that the example is a something which is executed in the terminal (as opposed to being a code sample) and usually these are not supposed to be entered when you execute a command. You'll see the `$` notation used throughout this learning experience.

Read the Manual!

The most important command in all of terminal life is `man`. Short for manual, the man command gives you information about how to use any given command. Whenever you encounter a new command, try looking at the manual entry.

Try typing the following into your command line:

`$ man echo`

What do you see? Is it clear what `echo` does?

`echo` simply prints whatever **arguments** you provided back to the terminals**standard output stream**. Type q to exit the manual.

Try this:

`$ echo hello world`

In the command `man echo`, `man` is the command and `echo` is the argument.

In the command `echo hello world` echo is the command, and the text "hello world" is the argument.

> Pro-tip: www.explainshell.com is a great resource for understanding shell commands. Commands can get complex, take a look at this command which copies any file under `/path/to/search/` whose name contains the word "smile" to the directory `/target/path/`:
>
> `find /path/to/search/ -type f -name ".*smile.*" | xargs cp -t /target/path/;`
>
> Read about this command on Stack Overflow then try examining that command on explain shell. The command line is very powerful. Don't worry if this command doesn't make sense yet.

Current Working Directory

The file structure you see in the Terminal is the same as the one you see in the `Finder` application. Finder tends to hide some of the folders from you to keep things simple for most users, but everywhere that you go in Finder is accessible through 'Terminal'.

Where am I?

Typically the shell will start in your `HOME` directory, each user has their own `HOME` directory, but on your computer it is common for you to be the only real user. At any given time a terminal shell process has one **current working directory**. Lets use the pwd (short for **print working directory**) command to show your current working directory:

Try This:

```
$ pwd
/Users/eschoppik
```

For Elie this is `Users/eschoppik`, what is the **current working directory** of your shell process?

> Pro-tip: the tilde character (`~`) is mapped (or ailiased) to the `HOME` directory on most *nix shells. Try: `echo ~`, is this the same as the output you got from `pwd`?

Try This:

`$ open .`

Wherever we are, `open .`, opens a `Finder` window in the current directory, this can be handy sometimes.

> Pro-tip: the `.` character can be used as a reference to the**current working directory** in the terminal.

Looking Around

What can we find out about the **current working directory** ?

One of the most useful commands is:

`$ ls`

Which lists the files and directories in the current working directory. Personally I find this a little difficult to read so I use the long form by envoking the `-l` option:

```
$ ls -l

  total 48
  drwxr-xr-x    2 eschoppik  staff    68 Dec  4 15:13 Applications
  drwx------+   6 eschoppik  staff   204 Mar 23 18:20 Desktop
  drwx------+  11 eschoppik  staff   374 Feb 27 10:57 Documents
  drwx------+ 141 eschoppik  staff  4794 Apr  5 08:04 Downloads
  drwxr-xr-x    3 eschoppik  staff   102 Nov 12 13:56 Justinmind
  drwx------@  56 eschoppik  staff  1904 Apr  4 21:58 Library
  drwx------+   3 eschoppik  staff   102 Nov  4 10:49 Movies
  drwx------+   8 eschoppik  staff   272 Mar  5 15:48 Music
  drwx------+  20 eschoppik  staff   680 Mar 23 12:53 Pictures
  drwxr-xr-x+   5 eschoppik  staff   170 Nov  4 10:49 Public
  drwxr-xr-x    3 eschoppik  staff   102 Jan 31 13:21 bin
  -rwxr-xr-x    1 eschoppik  staff   184 Nov  8 16:41 git_profile.sh
  -rw-r--r--    1 eschoppik  staff   327 Mar 27 09:22 gitshell.sh
  drwxr-xr-x   22 eschoppik  staff   748 Feb  3 15:15 hashes
  drwxr-xr-x    3 eschoppik  staff   102 Apr  1 10:34 helloroom
  -rwxr-xr-x    1 eschoppik  staff   409 Nov 15 12:13 phpshell.sh
  -rwxr-xr-x    1 eschoppik  staff   299 Jan 31 13:27 rorshell.sh
  -rwxr-xr-x    1 eschoppik  staff   316 Feb  2 10:35 rorshellws.sh
  lrwxr-xr-x    1 eschoppik  staff     5 Nov  7 18:22 work -> /work
```

Now I can see a lot more clearly what files are in my current working directory. Some of these items are files, some are directories and in my case also have a `link` which we'll deal with on another day :)

The `ls` command can take a directory as an argument. This lists the content of the provided directory:

```
$ ls -l Documents/

  total 40360
  drwxr-xr-x   4 eschoppik  staff      136 Feb 22 20:01 Rails
  -rw-r--r--@  1 eschoppik  staff  8154896 Feb 27 10:57 Profile.png
  -rw-r--r--@  1 eschoppik  staff  6258658 Feb 27 10:57 Profile2.png
```

The ls command can also take a wildcard (`*`) as an argument. This only lists items in Documents/ that end with .png

```
$ ls -l Documents/*.png

-rw-r--r--@ 1 eschoppik  staff  8154896 Feb 27 10:57 Documents/Profile.png
-rw-r--r--@ 1 eschoppik  staff  6258658 Feb 27 10:57 Documents/Profile2.png
```

Hidden Files

Have you ever heard of `hidden files` ? Well it's true, they are real! and we can see them by envoking the `-a` option:

```
$ ls -la

total 368
  drwxr-xr-x+  76 eschoppik  staff   2584 Apr  6 10:30 .
  drwxr-xr-x    6 root    admin    204 Nov  4 10:47 ..
  -rw-r--r--@   1 eschoppik  staff  15364 Apr  2 16:00 .DS_Store
  -rw-------    1 eschoppik  staff   8949 Apr  1 17:21 .bash_history
  -rw-r--r--    1 eschoppik  staff    285 Mar 17 14:50 .bash_profile
  -rw-r--r--    1 eschoppik  staff     59 Feb  2 13:47 .bashrc
  drwxr-xr-x    5 eschoppik  staff    170 Dec  5 13:21 .bundler
  -rw-r--r--    1 eschoppik  staff    379 Mar  3 17:36 .gitconfig
  drwxr-xr-x   30 eschoppik  staff   1020 Feb  2 13:47 .rvm
  drwxr-xr-x    2 eschoppik  staff     68 Dec  4 15:13 Applications
  drwx------+   6 eschoppik  staff    204 Mar 23 18:20 Desktop
  drwx------+  11 eschoppik  staff    374 Feb 27 10:57 Documents
  drwx------+ 141 eschoppik  staff   4794 Apr  5 08:04 Downloads
  drwxr-xr-x    3 eschoppik  staff    102 Nov 12 13:56 Justinmind
  drwx------@  56 eschoppik  staff   1904 Apr  4 21:58 Library
  drwx------+   3 eschoppik  staff    102 Nov  4 10:49 Movies
  drwx------+   8 eschoppik  staff    272 Mar  5 15:48 Music
  drwx------+  20 eschoppik  staff    680 Mar 23 12:53 Pictures
  drwxr-xr-x+   5 eschoppik  staff    170 Nov  4 10:49 Public
  drwxr-xr-x    3 eschoppik  staff    102 Jan 31 13:21 bin
  ......More Files.....
  -rwxr-xr-x    1 eschoppik  staff    184 Nov  8 16:41 git_profile.sh
  -rw-r--r--    1 eschoppik  staff    327 Mar 27 09:22 gitshell.sh
  drwxr-xr-x   22 eschoppik  staff    748 Feb  3 15:15 hashes
  drwxr-xr-x    3 eschoppik  staff    102 Apr  1 10:34 helloroom
  -rwxr-xr-x    1 eschoppik  staff    409 Nov 15 12:13 phpshell.sh
```

```
-rwxr-xr-x    1 eschoppik  staff    299 Jan 31 13:27 rorshell.sh
-rwxr-xr-x    1 eschoppik  staff    316 Feb  2 10:35 rorshellws.sh
lrwxr-xr-x    1 eschoppik  staff      5 Nov  7 18:22 work -> /wor
```

Hidden Files are typically used by applications to store configurations and there will be a many of them in your home directory. Most users don't want to be editing these files so they don't show up in `Finder`, but you as a software developer will be editing some these for yourself later on in the course.

Hidden files are hidden because their names begin with `.`

> Pro-tip: `ls` las a LOT of options. Try looking at the manual entry by using `man ls`.

Mini Review - Current Working Directory

- pwd
- Home Directory
- open .
- ls -la

Navigating Around

Root Directory

Another important directory is the root directory `/`

Try This:

```
$ cd /
$ pwd
```

The files on your computer are structured in a tree. The 'top' of the file system is know as the `root` directory. That may sound upside down, but in our case the root is at the top.

We can move to the **root directory** with the command `cd /`.
We can move back to your **home directory** with the command `cd ~`.

```
$ cd ~
$ pwd
/Users/eschoppik
```

Remember, the `~` always refers to the current user's home directory, this is handy for scripts and for you, but you can use the full path just as well if you know it, `pwd` will give you the full path.

Relative Paths

Try this:

```
cd ../
pwd
```

What happened? Which directory are you in?

In the terminal, the `.` character refers to the **current working directory** and two dots `..` refers to the current directories **parent** directory. What happens if you try this:

```
$ cd /
$ cd ..
```

The terminal ignores `cd ..` in this case. the root directory is the only directory in your entire filesystem that does not have a parent.

(`../`) is a **relative paths** and you can use it anywhere you would use a path. What happens if we type:

```
$ ls -l ~/Documents/../

total 0
drwxr-xr-x+ 11 Guest  _guest  374 Nov  4 10:47 .
drwxr-xr-x   6 root   admin   204 Nov  4 10:47 ..
drwx------+  3 Guest  _guest  102 Nov  4 10:47 Desktop
drwx------+  3 Guest  _guest  102 Nov  4 10:47 Documents
drwx------+  4 Guest  _guest  136 Nov  4 10:47 Downloads
drwx------+ 26 Guest  _guest  884 Nov  4 10:47 Library
drwx------+  3 Guest  _guest  102 Nov  4 10:47 Movies
drwx------+  3 Guest  _guest  102 Nov  4 10:47 Music
drwx------+  3 Guest  _guest  102 Nov  4 10:47 Pictures
drwxr-xr-x+  4 Guest  _guest  136 Nov  4 10:47 Public
```

The command means, list the contents of the parent of `~/Documents/` So it listed the contents of `~`, or the home directory.

Any path starting with a `/` is said to be an **absolute path** and it is the complete path starting from the root directory. Relative paths (ones that do not begin with a `/`) are relative to your current location.

Tab Completion

Hitting `<TAB>` autocompletes. Hit `<TAB>` constantly. Try it right now! Type:

`$ cd ~/L` THEN HIT TAB!

This trick will save you so much time. Here's another trick, type:

`$ cd ~/` now DOUBLE TAP TAB. What happened?

This way you can easily see the competing outcomes of autocomplete. What happens if you type:

`$ cd ~/D` then double tap tab?

The competing options for me are `Desktop/` , `Documents/` , and `Downloads/`

Pair Practice

Exercise: 5 minutes in Pairs

1. Using Finder: Pick a directory somewhere under the /Users directory on your partner's computer
2. Your Task: Navigate to that directory in a single command from your home directory using a relative or absolute path
3. Help your partner if they are having trouble and use Tab Completion

Mini Review - Navigating Around

- root directory `/`
- `../`
- Absolute and Relative Paths
- Tab Completion

File Manipulation

mkdir

Now that we know how to move around, it's time to make some changes. We can make directories with the `mkdir` command. Look at the man page by using the command `man mkdir`. What's the format of the command for making a directory?

```
MKDIR(1)                  BSD General Commands Manual                  MKDIR(1)

NAME
     mkdir -- make directories

SYNOPSIS
     mkdir [-pv] [-m mode] directory_name ...

DESCRIPTION
     The mkdir utility creates the directories named as operands, in the order spe
     mode rwxrwxrwx (0777) as modified by the current umask(2).
```

Operands (or arguments or parameters) are what comes after a command, so we write `mkdir living_room` to make a new room, where we will keep our couches. Keep your directory names lowercase in almost every case. Separating words with underscores is called snake_case.

> Pro-tip: WordsLikeThis are called CamelCase. Programmers frequently argue about snake_case and CamelCase

Try This:

```
$ cd ~
```

```
$ mkdir living_room
```

What command can you use to see the results of your handywork?

Adding and Editing Files

Let's `cd` into our new `living_room` Look around with `ls`, and `ls -la`. What do you see?

Exercise I want my living room to have a bookshelf full of books. Let's make a file that lists all of our books. Try this:

`$ touch books.txt`

Now try listing the contents of your current directory. What did the command `touch` do? You can use `touch` to do more than just create files. Try reading the man page for touch!

We've created a file, lets try editing text with the command line! Type:

`$ nano books.txt`

> If you get an error like `bash: nano: command not found` then try installing `nano` with homebrew by running the command `brew install nano`

Lets add some books to our text file. Copy and paste the section below so we all have some books in common, and save the file. Add some books of your own choosing as well! Make sure the books you add are in the same format: `<author_given_name>, <author_last_name>:<title>`.

```
Carroll, Lewis:Through the Looking-Glass
Shakespeare, William:Hamlet
Bartlett, John:Familiar Quotations
Mill, John :On Nature
London, Jack:John Barleycorn
Bunyan, John:Pilgrim's Progress, The
Defoe, Daniel:Robinson Crusoe
Mill, John Stuart:System of Logic, A
Milton, John:Paradise Lost
Johnson, Samuel:Lives of the Poets
Shakespeare, William:Julius Caesar
Mill, John Stuart:On Liberty
Bunyan, John:Saved by Grace
```

When you're done, exit nano by hitting `ctrl+x`. Nano will ask if you want to save, type `y` then hit enter. Now nano will ask where you want to save, and it will have auto-populated with books.txt. Just hit enter.

Now try `ls -la` again. Do you see the `books.txt` file? Look at the contents with:

`$ cat books.txt`

What does `cat` do?

There are other ways to view text files as well. Try

```
$ less books.txt
```

What does `less` do? Inside of your `less` window, try typing `/Mill` then hitting enter, what happened?

> Pro-tip: use `cat` when you have a short text file, and especially when you want the output of the text file to remain in your command prompt. Use `less` when you have lots of text to search through.

> Pro-tip: when you type `man command` you're using `less`. Try searching through man pages using the same /searchWord trick we used in `less`

echo and Redirection

Try This

```
$ echo "This bookshelf flexes under the weight of the books it holds."
```

Recall that `echo` just echoes (outputs) what we give to it as arguments (same as operands). Now we want to put that line in a file called `bookshelf.txt`.

Try This

```
$ echo "This bookshelf flexes under the weight of the books it holds" >
bookshelf.txt
```

Using the closing angle bracket `>` in this way is called **redirection**. Every command that we run in the shell has an input, an output, an error output, and arguments/operands. We are saying: "Run `echo` with this string as an operand, and take the output and put it in a new file called bookshelf.txt" Try running `ls` again, and `cat` our new file.

Try This

```
$ echo "Hmmm" > bookshelf.txt
```

Now `cat booshelf.txt` again. Our old text has been replaced with the new text "Hmmm". Sometimes we'll want to **append** to the existing text instead of overwriting it. We use two angle brackets `>>` to append the string to the end of the file:

Try This

```
$ echo "This bookshelf flexes under the weight of the books it holds" >
bookshelf.txt
$ echo "It does not break, it does its job admirably" >> bookshelf.txt
```

Try `cat bookshelf.txt` to see the result. The first command replaced "Hmm" with "This bookshelf flexes under the weight of the books it holds", the second command appended "It does not break, it does its job admirably" after the existing text.

> Pro-tip: You can use `&&` to execute another command if the first command succeeds. Try this single command version of what we just did:

```
$ echo "This bookshelf flexes under the weight of the books it holds" > bookshelf.
```

Piping

The Unix Philosophy is "do one thing, and do it well." Complex problems are solved by using small and simple modules, and chaining them together. This is a great way to think about software, and in terminal programming we chain commands using the `|` or pipe character.

Let's look back at our books. Read out the file. Notice that the list of books is unsorted! Lets organize this list using the `sort` command.

Pipes allow us to use the output from one command as the input for another command.

Try This

```
$ cat books.txt | sort
```

We took the output from `cat books.txt` and sent it through a pipe to `sort`. The output of `cat books.txt` becomes the input of `sort`. The output of `sort` printed to our screen. Now lets redirect the output of `sort` to a file:

Try This

```
$ cat books.txt | sort > sorted_books.txt
```

There are dozens of powerful tools we can leverage using pipes. One of the ones you'll be using the most is `grep`.

Try This

```
$ cat books.txt | grep Mil
```

See how we filtered out just the lines that contain Mil? Try grepping for something else.

Adapted from http://en.flossmanuals.net/command-line/piping/

Moving

Now that we have our books sorted, we really don't need our unsorted list of books. `mv` stands for move, and that's how we move files and folders from place to place.

Try This

```
$ mv sorted_books.txt books.txt
```

Examine the contents of our current directory. What has changed?

Copying

To copy files, we use the `cp` command. Extrapolate from the way we used `mv` to copy the file `bookshelf.txt` to add a file `second_bookshelf.txt`.

Try This

```
$ cp bookshelf.txt second_bookshelf.txt
```

What happend? What are the contents of second_bookshelf.txt?

Removing

`rm` is short for remove. Use `rm` to remove the `second_bookshelf.txt` file we just created with `cp`.

Try This

```
$ rm second_bookshelf.txt
```

> Pro-tip: `rm` does not send things to your trash can, it deletes them permanently. Be careful when using `rm`.

The "Recursive" Option

By default, commands like `cp` and `rm` only apply to the file specified. We can copy and remove entire directories with the `-r` option. `-r` stands for recursive, which is a very important term in computer programming. In this context it means "follow the directory structure through sub-directories until we are at a 'leaf node' in our directory tree."

Try This

```
cd ..
cp living_room study
```

We get an error: `cp: study/ is a directory (not copied).` To copy directories, we need to use `-r`:

```
cp -r living_room study
```

Now examine the contents of the directory 'study'. We copied all of our files to the new directory! Just like `cp`, `rm` will not work by default on a directory. Try `rm study` and you'll get the same error. Try this instead:

```
$ rm -r study
```

The study is gone. You can also use `rmdir` for this purpose.

Filename Wildcards

Sometimes we want to refer to a bunch of similar files, to do this we can use wildcards. The most common wildcard to use is `*` usually along with a file extension: **Try This**

```
$ ls -la *.txt
```

> Pro-tip: This is basically the same as `ls -la | grep .txt`. Can you think of a time when piping to `grep` would be preferable to a simple wildcard?

For more ideas go here: How to Use Wildcards

Mini Review - File Maniuplation

- `mkdir`
- editing files
- echo
- redirection `>` and `>>`
- piping
- moving, copying and removing
- recursive option

File Permissions

Lets examine our current working directory to discuss permissions.

`$ ls -l`

I'll cherry pick one line to describe permissions:

```
-rwxrw-r--  1 Tyler  staff  413 Oct 15 11:22 books.txt
```

The column on the left e.g.: `-rwxrw-r--` displays the files' permissions. That is whether or not you can read, write or execute the file. The first character is one of three:

- `-` for a regular file
- `d` for a directory
- `l` for a "link" which we'll talk about more another time.

The next 9 characters are one of 4 characters, and refer to what can be done to the file. These should be thought of in groups of 3, and they describe the permissions for different people/groups of people. So for our line: `-rwxrw-r--` we have the leading `-` telling us it's a file, then 3 groups:

- `rwx` The owner's permissions are first, the owner can read write and execute
- `rw-` The group's permissions are next, they can read and write
- `r--` Everyone else's permissions are last, everyone can read this file

After that we see `1 Tyler staff 413 Oct 15 11:22 books.txt`. This line tells that Tyler owns this file; the file belongs to the staff group; its size is 413 bytes; it was last modified Oct 15th at 11:22 and the name of the file is books.txt. The 1 at the start refers to how many files a directory contains, it is always 1 for regular files but might be larger for directories.

You can change permissions with `chmod` (short for change mode) and you can change file ownership with `chown` (short for change owner). For now we'll leave permissions at that if you're interested in more this is a nice tutorial:http://en.flossmanuals.net/command-line/permissions/

You can also look at the man pages for `chown` and `chmod`. Lets try changing the permissions and ownership of our bookshelf!

```
$ chmod 400 bookshelf.txt
$ ls -l bookshelf.txt
-r--------  1 Tyler   staff  106 Oct 15 13:40 bookshelf.txt
```

Now only the owner has permission to do anything, and all they can do is read the file. If you try to edit that file with nano, and save, what happens?

Lets try changing the ownership of the file:

```
$ chown StrangeUser:staff bookshelf.txt
chown: bookshelf.txt: Operation not permitted
```

This failed, because even though your user owns that file, you're not allowed to write to it! So, this begs the question -- if no one is allowed to write to this file, can we ever change or delete it?!

Enter `root`. Root is the administrative user. Root has all permissions. Root can do anything. You can become this "super user" to run a command using the `sudo` (super user do) command.

Try This:

```
sudo chown StrangeUser:staff bookshelf.txt
```

You should be asked for your password, then the command will execute as if you are `root`. Root has all permissions for all things. Running commands as `root` can be dangerous, and unless you know what you're doing and know why you need to be root, I suggest not using `sudo`.

> Pro-tip: You can use `!!` as a shortcut to repeat the last command. A common idiom is to try a command, and if permission is denied to your current user, try `sudo !!` to repeat the previous command as root.

History

Wow, we've done a lot of work. Remembering all these commands can be hard. Luckily our shell remembers a lot of what we've done for us! Try tapping the up arrow in your shell. What happens? We can scroll up and down through the most recent commands we've executed.

Try This

`$ history`

and

`$ history | grep cd`

Searching through history can be very useful if you know you've done something, but can't remember exactly how you did it. You can also combine the power of `history` with auto-complete. Try hitting `ctrl+r` then typing `ls`. What happens?

You can scroll up and down through all recently used commands that contain the string 'ls' using `ctrl+r` and `ctrl+shift+r` to go backwards.

If you don't want to execute any of these commands, type `ctrl+c`. Control+c is a powerful command that you can use at any time to kill the currently running terminal process, or exit many terminal applications.

Review

Getting Help

Don't forget about the command `man`! Short for **manual**, it will give a (hopefully) detailed

explanation of that command. Sometimes that explanation will be too detailed for you. When you get lost in a man page and you want to understand it, start again from the beginning of the **man page** and keep repeating. Hopefully you will get further into the page each time you read it.

> Pro-tip: when you're feeling meta, try the command `man man`

Many advanced commands also accept the --help, or -h option, but not all, but if you get stuck it can be worth a try. Most of the commands covered in this simple overview do not support this feature

```
$ git --help
```

Terminal Cheat Sheet

Bookmark this:

- http://bit.ly/terminalcheats

Homework!

Solve the Command Line Murder Mystery. To get started, fork and clone the repo to your local machine. Further instructions can be found in the README for the mystery. Good luck!

Additional Resources

Learn the Command Line the Hard Way is a great book for learning the command line. Check it out! http://cli.learncodethehardway.org/book/

New ZPD Response