Vim class notes

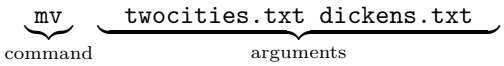
Chris Wallace

Week 2

First, we talked about what went well last week and what didn't. Namely, we discussed how there are students of all levels here so we're going to go slow, and fast, by having a project to work on soon. Additionally, we noted that the easiest way to log into cloud9 was to visit `https://c9.io/login` directly.

File and Folder Manipulation

To use commands, we first type the command name, followed by any *arguments* that we want the command to work on:



We introduced the following commands for working with *files*:

ls	list a directory's contents.
cat	put files together and print them. We can use this to look at the contents of a file.
mv	move or rename a file.
cp	copy a file.
rm	remove a file.
touch	create an empty file.

We also introduced the following commands for working with *folders*:

cd	or *change directory*. We use this to move around folders.
mkdir	makes a directory.
rmdir	removes a directory, although only when it's empty.

When moving around folders with `cd`, we introduced the concept of *absolute* and *relative* paths. Absolute paths are places described as where they are from the root directory, or `/`. An example would be `/home/cwwllac/Desktop/class`. We also discussed how a user's home directory is pretty important, so we get a shortcut to it: the squiggly tilde: (~). An example path using the tilde (which is still an absolute path) is `~/Desktop/class`

Relative paths are simpler, they are just getting you somewhere *relative* to where you are now. If you were home, to get to `Desktop/class` you could simply do: `cd Desktop/class`. The two shortcuts to note when using relative paths are (`.`) and (`..`) - a single period is the *current* folder we're in, and a double period is the folder *above* (or "back behind") this one.

Finally, we talked about how to escape a "stuck program" with Control+C.

Input Redirection

First, we downloaded a file using the following command:

```
wget scdojo.s3.amazonaws.com/vim/twocities.txt
```

This file happens to be the book *A Tale of Two Cities* by Charles Dickens. We played around with this book with some commands. Note that any line beginning with a `#` is a comment and is not code.

```
# count the number of lines, words, and characters in twocities.txt
wc twocities.txt
# search twocities.txt for any line containing the word "times"
grep "times" twocities.txt

# redirect the output of a command to a file (times.txt)
grep "times" twocities.txt > times.txt
# count the number of lines, words, and characters in that new file
wc times.txt

# redirect the output of a command to another command
grep "times" twocities.txt | wc
```

We then installed a new program called *cowsay*:

```
# update package lists. We use sudo because this requires being "super user"- the administrator
sudo apt-get update
# install cowsay
sudo apt-get install cowsay

# say hi
echo "hello world"
# say hi using a cow
echo "hello world" | /usr/games/cowsay
# save all that to a file
echo "hello world" | /usr/games/cowsay > hello_cow.txt
```

Put it in a File

Finally, using vim, we put these things together in an actual file:

1. Open vim with `vim hello_world.sh` - we ended it in .sh because this is a *shell script*
2. Press **i** to go into insert mode
3. Add a comment to the file (line starting with `#`) about what this file will do
4. Add any script you want using `echo`, `cowsay`, etc

We also introduced variables, which are things that can hold value and even have their value change. See a full example below:

```
# script to say hi and ask your name
echo "Hello world!"
name="Chris"
echo "My name is $name", what is yours? (please type)"
read name
echo "Cool! Hello there $name"
```