# *csce215 — UNIX/Linux Fundamentals*
# *Fall 2021 — Lab Assignment 2*

*This assignment is intended to provide some practice and additional content for the material covered in lecture on Monday, August 30. You'll create some directories and some files, practice using the `vim` text editor, and gain experience creating, moving, renaming and deleting files. The assignment is meant to be completed in the lab sessions on Wednesday, September 1, and must be submitted by 11:59pm on Friday, September 3. A total of 40 points are available.*

## 1 Sign in, start a terminal, and start recording

Just like for Lab 1, you'll be using a terminal window and submitting a recording created using the `asciinema` tool. If you've forgotten how to do this, refer back to the notes or lab assignment for week 1. You'll want to choose a different filename this time, but you can choose the filename for yourself. (Perhaps `lab2.cast`?)

As a reminder, if your `asciinema` session ends for some reason (closing the terminal, typing `exit`, pressing `Ctrl-D`, etc.), you'll need to resume the recording before continuing. For example, if your recording file is `lab2.cast` in your home directory, then restarting the recording would look like this:

```
$ asciinema rec --append ~/lab2.cast
asciinema: appending to asciicast at /home/jokane/lab2.cast
asciinema: press <ctrl-d> or type "exit" when you're done
```

Here's one more reminder of an important fact:

> We can only award points for things that are recorded into the screencast file that you submit.

Sign in to the lab computer, open a terminal, and use the `asciinema rec` command to begin a recording.

## 2 *Setting up a* `215` *directory*

Now that we know how to create directories, let's start to get things organized. One reasonably good idea is to create a different directory for each of your courses. For this course, let's use a directory called `215`, with a subdirectory for each lab assignment. So you'll do all of the work for this lab inside the directory `/215/lab2`.

***If you already have a 215 directory*** A few of you might already have a `215` directory. If so, you'll want to rename the existing one, so it does not interfere with your work this semester. Fortunately, we know how to rename directories; a command like this should work:

```
$ mv 215 old-215
```

If, like most students in the class, you don't already have a `215` directory, you can ignore this `mv` step and proceed directly to using `mkdir`.

Use `mkdir` to create a subdirectory called `215` within your home directory. Within `215`, create subdirectories called `lab1` and `lab2`. Then use `cd` to navigate to the `lab2` directory. *3 points*

## 3 *Downloading and decompressing some example files*

Soon, you'll practice organizing files by separating some text files into different subdirectories based on their contents. First, you'll need to download a zip archive containing the files that we'll work with. There's a nice command called `wget` that can download a file for us.

**wget** 

Download a file from a URL.

In this case, the URL of the file we want is

`https://cse.sc.edu/~jokane/215/wilsons.zip`

So the command

```
$ wget https://cse.sc.edu/~jokane/215/wilsons.zip
```

will download the file to the current directory.[1] If this has worked correctly, you'll see a progress bar as the download happens, followed by a message saying that `wilsons.zip` has been saved.

Within your `lab2` directory, use `wget` to download the file `wilsons.zip`. Use `ls` to see the file. *3 points*

Once the file is downloaded, we'll want to extract all of the files stored in the zip file. There's a command to do that: `unzip`.

## unzip

Extract files from a Zip archive.

`-l` (dash ell) Don't extract. Just list the files instead.

So use the command

```
$ unzip wilsons.zip
```

to create a subdirectory called `wilsons` in your `lab2` directory, containing a collection of various text files. If you've done this correctly, `unzip` will tell you that it is 'inflating' several different files. When it's finished, use `ls` to take a look at the files that were extracted.

Use `unzip` on the downloaded zip file, to extract all of the sample files it contains. Then execute an appropriate `ls` command to see a list of all the extracted files. *3 points*

[1] Did you paste the URL from this PDF into your terminal and get a message saying `ERROR 404: Not Found`? If so, two things: (1) Remember the rule in this course about not pasting things into the terminal? Just checking. (2) Copying from a PDF document does not always give the expected results. Perhaps in this case, the `~` character did not copy correctly? For reasons (1) and (2) aforementioned, you should type the URL for yourself. It's not *that* long.

## 4 *Separating the Wilsons*

You'll notice a collection of text files written by famous people named Wilson here. Unfortunately, files written by three different people with the surname Wilson have been mixed together:

1. Twentieth century Canadian geologist **Alice Wilson**,
2. American composer **Brian Wilson** of the Beach Boys, whose career spans 1961 to the present, and
3. Nineteenth century English professor, philosopher, and logician **Cook Wilson**.

Your job in this section is:

1. to put all of the files written by Alice Wilson in a subdirectory called `alice`,
2. to put the files written by Brian Wilson in a subdirectory called `brian`, and
3. to put the files written by Cook Wilson in a subdirectory called `cook`.

(The difference between text about geology, text about philosophy, and 60's-/70's-era song lyrics should be pretty obvious, but if you are unsure, please ask!) When that's finished, get rid of the original `wilsons.zip` and the `wilsons` subdirectory, which should be empty.

This task will give you a chance to use several of the commands that we learned over the last two weeks, including `mkdir` to create the directories; `less` (or `cat` or even `vim`) to see the files and determine who the author is; `mv` to move them into place; `rm` to delete `wilsons.zip`; and `rmdir` to remove the `wilsons` subdirectory when it's empty.

Create subdirectories called `alice`, `brian`, and `cook`. *3 points*

Use the command of your choice to view each of the text files in `wilsons`. Move each one to either `alice`, `brian`, or `cook`, as appropriate.

(There are ten text files. We'll award one point for viewing and moving each one correctly.) *10 points*

Use `rm` to delete `wilsons.zip`. *2 points*

Use `rmdir` to delete the `wilsons` subdirectory. *2 points*

Finally, let's check the results. Because the files we're working with are spread across several directories, the usual form of `ls` command isn't as helpful as we'd like. However, we can instead use the `find` command, which goes through an entire tree of subdirectories, showing us everything along the way.

**`find`** 

Look for files that meet certain criteria (default: all files and directories under the current directory) and do things with them (default: print their names).

We'll see `find` in more detail later in the semester, but for now, you can just use it without any parameters to show everything. If you've done this correctly, you'll see a list that shows each of the subdirectories you created and each of the ten text files inside them. As a hint, you should see three files in the `alice` directory, four in `brian`, and three in `cook`.

Use `find` to list all of the files below your `lab2` directory. *2 points*

## 5 *Prime time for editing*

Now let's shift gears and practice editing a file using `vim`. Use `wget` to download the file at this URL:

```
http://cse.sc.edu/~jokane/215/primes.txt
```

This file is *supposed* to be a nicely-formatted list of prime numbers less than 500. But as you can see by looking at the file, it has some problems. Let's use `vim` to correct the mistakes. Open the file in `vim` using this command:

```
$ vim primes.txt
```

Most of the corrections can be done readily using the vim techniques discussed in the notes.

1. The number 5 is, for some reason, incorrectly written out as 'five'. Delete this word and insert the numeral '5' in its place.
2. Some composite numbers have sneaked in. The only prime number between 110 and 120 is 113. Delete the other numbers in this range.
3. Some prime numbers are missing. Insert 389 and 503 in their appropriate places.
4. Two of the lines are out of order. Swap them into the correct order.
5. The list is only supposed to go up to 499, but continues after that. Delete all of the numbers greater than 500.
6. There's some strange text on the second and third lines, between 79 and 83. Delete it.

When you've finished these steps, save the file, quit vim and use `cat primes.txt` to display the completed file in your terminal recording. This is how we'll check the correctness of your edits; we won't be able to give points for editing the file if you don't cat it when you've finished. As an example, the first few lines of your file might look like this:

```
0 1 2 3 5 7 11 13 17 19 23 29 31 37
41 43 47 53 59 61 67 71 73 79
83 89 97 101 103 107
```

Use vim to edit primes.txt, until it contains a correct, ordered list of prime numbers less than 500. Then exit vim and use cat to display the results. *8 points*

Now we have a correct list of prime numbers up to 500, but the file probably looks ugly because the length of the lines are not even. As a last step, let's fix this formatting, so that every line is the same length. Along the way, we'll see some very quick glimpses into what vim can really do.

1. Open the file in vim again.

2. From normal mode, give this command to tell vim how long we like to have lines wrapped to:

   ```
   :set textwidth=40
   ```

   Vim has hundreds of options like this, to control many different aspects of how it works. If you are interested in customizing vim more in the future, you can see a list of options in vim's built-in help, with the command

   ```
   :help option-list
   ```

   And, in general, the :help command in vim can be a great way to learn more.

3. Now we can reformat the lines to get each one to be the same length. (Suggestion: Read these next few paragraphs carefully before trying this.) The command to do this is gq in normal mode *(with no colon)*. After you type gq, vim waits for a *movement* command, and formats the lines that you moved over.

   As an example, suppose you start at the top of the file and use the command gqj. The gq is the line re-wrapping command that we want to use, and j represents a movement command to go down one line. (We also could have used gq followed by pressing the down arrow key.) In this case, the cursor would move from the first line to the second line, so the gq command would apply to those two lines; only those first two lines would be wrapped so that they fit nicely within the textwidth limit set above.

   In this case, we want to reformat the entire file, so you should go to the top of the file in normal mode, type gq, and then give a single movement command to go to the end of the file. (Check the notes if you've forgotten how to jump to the end of the file.)

   The final result should look something like this:

   ```
   0 1 2 3 5 7 11 13 17 19 23 29 31 37 41
   43 47 53 59 61 67 71 73 79 83 89 97 101
   103 107 109 113 127 131 137 139 149 151
   157 163 167 173 179 181 191 193 197 199
   211 223 227 229 233 239 241 251 257 263
   269 271 277 281 283 293 307 311 313 317
   331 337 347 349 353 359 367 373 379 383
   389 397 401 409 419 421 431 433 439 443
   449 457 461 463 467 479 487 491 499
   ```

This sort of arrangement, in which a vim command affects the text covered by a movement command that comes after, is a very common pattern in vim. These kinds of combinable commands give vim much of the 'power' that vim users like to emphasize.

In vim, set the textwidth option to 40. *1 point*

Use the gq command to re-wrap the lines in primes.txt so that each has at most 40 characters. Then save and quit vim. Use cat to show the full result in the recording. *3 points*

## 6 *Wrapping up*

Now that we've finished the lab, it's time to wrap up. This will work just like in Lab 1.

- Use Ctrl-D or exit to conclude the terminal recording to lab2.cast. Check for the message saying:

```
asciinema: recording finished
asciinema: asciicast saved to lab2.cast
```

- Consider replaying the recording, to ensure that each of the , tasks are shown being completed. One command to do this is:

```
$ asciinema play -i 0.5 lab2.cast
```

  Here, the -i 0.5 means to eliminate all delays longer than 0.5 seconds.

- Upload your completed lab2.cast to the Lab 2 submission site at the dropbox site.

  https://dropbox.cse.sc.edu/

- Sign out of the lab computer.