csce215 — UNIX/Linux Fundamentals
Spring 2022 — Assignment 2

This assignment is intended to provide some practice and additional content for the material covered in lecture on Monday, January 24. 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 started in the lab sessions on Wednesday, January 26 and Thursday, January 27. It must be submitted by 11:59pm on Sunday, January 30. A total of 92 points are available.*

1 Before you begin

Tell us you're here As usual, please remember to give yourself credit for attending the lab session, by scanning the QR code or entering the password at the dropbox site.

Record your attendance.

If you don't finish during the lab session This assignment is intended to be started during your scheduled lab time on Wednesday or Thursday. Some students may complete it during that time, but many of you may need to complete the assignment at some later time. Here are some quick facts about returning to finish the assignments after the scheduled lab time:

- You can return to the lab any time, except for the few instances when a class is meeting in the room. Check the course website for a specific schedule.
- To gain access to the lab, you may need to use the door combination, which was distributed on the first day of class, posted in the course Slack, and also available by signing in here:

  `https://cse.sc.edu/resources/cse-linux-workstations`

  (The combination can't be included here, because this document is posted on a public website.)

- Several times throughout the week are designated as 'office hours', meaning that a TA is scheduled to be present in the lab and available to help with any questions you may have. These times are listed in the syllabus.
- When you return, simply start a new `recbash` recording and continue from where you left off. Then submit both recordings (or all of) the recordings when you've finished.
- The assignment should be submitted by 11:59 on Sunday night.

Don't forget the notes Many students will find it helpful to keep a copy of the lecture notes —both from Chapter 1 and Chapter 2— handy.

2 Sign in, start a terminal, and start recording

Just like for Assignment 1, you'll be using a terminal window and submitting a recording created using the `recbash` tool. Recall that the command to start a recording is

```
/class/215/recbash
```

Don't forget to check for the `[.]` in your command prompt to verify that the recording is working properly.

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 `recbash` command to begin a recording.

3 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 assignment. So you'll do all of the work for this assignment inside the directory `~/215/assignment2`.

If you already have a 215 directory A few of you might already have a `215` directory, for example if you've taken this course in the past. 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 `assignment1` and `assignment2`. Then use `cd` to navigate to the `assignment2` directory. *7 points*

4 *Copying the example files*

Soon, you'll practice organizing files by separating some text files into different subdirectories based on their contents. But first, you'll need to make a copy of those example files from their home, which is a directory called `/class/215/assignment2/wilsons`.

You'll want to copy that entire directory to your `assignment2` directory. To do this, first make sure that the `assignment2` directory you created is indeed your current directory. (Remember that `pwd` will show you the current directory.) Then issue a copy command, to copy `/class/215/assignment2/wilsons` to your current directory:

```
cp -r -v /class/215/assignment2/wilsons .
```

Recall that `-r` means to copy recursively, i.e. to go into the `wilsons` directory and copy all of the files and subdirectories in it, and that `-v` means 'verbose', i.e. to print out a line explaining each copy that's being made. And, of course, `.` refers to the current directory, which is the destination in this case. When `cp` is finished, use `ls` to take a look at the files that were extracted.

Within your `assignment2` directory, use `cp` to copy the `wilsons` directory into your `assignment2` directory. Use `ls` to see the files in `assignment2/wilsons`. *7 points*

5 *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 of `assignment2` called `alice`,
2. to put the files written by Brian Wilson in a subdirectory of `assignment2` called `brian`, and
3. to put the files written by Cook Wilson in a subdirectory of `assignment2` 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` 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; and `rmdir` to remove the `wilsons` subdirectory when it's empty.

Use `mkdir` to create subdirectories called `alice`, `brian`, and `cook` within your `assignment2` directory. *7 points*

Use the command of your choice to view each of the text files in `wilsons`. Use `mv` to move each one to either `alice`, `brian`, or `cook`, as appropriate. *25 points*

Use `rmdir` to delete the `wilsons` subdirectory. *5 points*

Finally, let's check the results, using a separate `ls` command on each of `alice`, `brian`, and `cook` to see the files in each one. As a hint, you should see three files in the `alice` directory, four in `brian`, and three in `cook`.

Use `ls` three times to show that `alice`, `brian`, and `cook` contain 3, 4, and 3 files, respectively. *5 points*

6 *Prime time for editing*

Now let's shift gears and practice editing a file using `vim`.

There is a file called `primes.txt` in the directory `/class/215/assignment2`. Use `cp` to copy this file to your `assignment2` directory. Include the `-v` option to `cp` to make it easier to see what is being copied.

Copy `primes.txt` into your `assignment2` directory, with the verbose option to `cp`. *7 points*

This file is *supposed* to be a nicely-formatted list of prime numbers less than 600. 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.[1]

[1] ...including 110 and 120, which because they are even and greater than 2, are obviously not prime numbers.

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 599, but continues after that. Delete all of the numbers greater than 600.
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:

```
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
```

Use `vim` to edit `primes.txt`, until it contains a correct, ordered list of prime numbers less than 600. Then exit `vim` and use `cat` to display the results. *18 points*

Now we have a correct list of prime numbers up to 600, 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
   ```

   Since this is a 'colon command' in `vim`, you'll need to press Enter to finish it. 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. If you've opened the `:help`, use `:q` to quit that buffer when you've finished.

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're not sure how to jump to the end of the file.)

   The final result should look exactly like this:

```
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
457 461 463 467 479 487 491 499 503 509
521 523 541 547 557 563 569 571 577 587
593 599
```

   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. *4 points*

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. *7 points*

7 *Wrapping up*

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

- Use `Ctrl-D` or `exit` to conclude the terminal recording. Check for the message saying something like this to confirm that the recording has been finalized:

```
Recorded to /acct/jokane/.recbash/2022-01-10_22:00:07_2164928.zip.
```

- Use this command

```
nautilus ~/.recbash
```

  to peruse your recordings. For the one(s) that you plan to submit, take a look at the `transcript.txt` inside the zip file to verify that your work has been recorded properly. Remember that there are no penalties for learning along the way; we'll check only for the existence of the correct answers in your transcripts. (If you need to continue using the terminal after using it to start `nautilus`, you may need to use `Ctrl-C` to terminate `nautilus` and return to the shell prompt.)

- Upload the zip file(s) of your work to the Assignment 2 submission station at the dropbox site, by dragging them from the `nautilus` window into the submission form.

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

- Sign out of the lab computer.