CS35L Fall 2022 Week 0 Linux Worksheet Solutions

1. Which text editors/IDEs have you used before Emacs? What are the differences between using them and Emacs? What's the most difficult part of using Emacs so far?
    a. *Homework hint!* Struggling to remember all the commands? Print this cheatsheet out (or have it handy on a screen nearby) as you do the lab. You do not need to have the most optimal use of keybindings, but do your best to minimize inputs.
    b. Don't forget to run `M-x open-dribble-file` for each editing session! (You will NOT need to do this for later homework assignments.)

    Answers may vary based on personal preferences. If you were not at discussion, take a few minutes to think about this.

2. Can you ssh into `lnxsrv11.seas.ucla.edu`?
    a. You need to be connected to the UCLA VPN.

    Hopefully you can! Make sure you're connected to the UCLA VPN. Come to office hours if you're still having trouble.

3. The command `env` lists all the set environment variables in your shell. What are some environment variables you see besides PATH?

    Answers will vary, but some that I see on SEASnet include:

    ```
    HOSTNAME=lnxsrv11.seas.ucla.edu
    TERM=xterm-256color
    SHELL=/bin/bash
    SSH_TTY=/dev/pts/14
    USER=sinnoh
    PATH=/usr/local/cs/bin:/usr/lib64/qt-3.3/bin:/u/cs/ugrad/sinnoh/perl5/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:/usr/local/cs/bin:/u/cs/ugrad/sinnoh/bin
    PWD=/u/cs/ugrad/sinnoh
    HOME=/u/cs/ugrad/sinnoh
    ```

    The main commonality of environment variables is that they contain configuration information about your system like what shell and terminal you're using, what your home directory is, and etc. Environment variables are also very useful for configuring web applications, and you can keep your configuration separate from your code!

4. Dr. Eggert asks you to complete Assignment 1 on GNU/Linux servers lnxsrv11, lnxsrv12, lnxsrv13, or lnxsrv15, with /usr/local/cs/bin prepended to your PATH.
    a. What is the effect of `export PATH=/usr/local/cs/bin:$PATH`?
    b. How do you "permanently" prepend /usr/local/cs/bin to your path?

    This command prepends /usr/local/cs/bin to your PATH. This means that any programs or commands you run will first be looked for inside the usr/local/cs/bin directory. You can "permanently" prepend /usr/local/cs/bin by editing the .config or equivalent file of whichever shell language you are running in the terminal.

5. Run the command `man man`. What does it say?
    a. You can navigate the man page by using the arrow keys to move up/down a line. 'd' goes Down one page, 'u' goes Up one page,'q' Quits
    b. Homework hint! Programs you'll want to man include:
        i. `which`
        ii. `wget`
        iii. `cp`
        iv. `ls`
        v. `chmod`
        vi. `find`
    c. Try using commands such as 'j', 'k', 'G', and 'gg'… Feeling suspicious?
    d. Are man pages too long for you? There are also TL;DR pages.

    Man man should say something like: "man - format and display the on-line manual pages". Note that man uses suspiciously similar keyboard inputs as vim. Take some time to read through the man pages for some of the commands!

1. Execute each of the following commands, and then describe what each one did: (Hint: `ls` is your friend!)
    a. `cd ~/Desktop`

    Changes directory to Desktop

    b. `mkdir foobar`

    Creates a directory named foobar inside current directory

    c. `cd foobar`

    Changes directory to foobar

    d. `touch silent.txt`

    Creates an empty file named silent.txt

    e. `echo "woof" > dog`

    Outputs woof to a file named dog

    f. `echo "oink" > pig.animal`

    Outputs oink to a file named pig.animal

g. `cat silent.txt`

Silent.txt is empty, so the output is empty

h. `cat dog`

Outputs content of dog, which is woof

i. `cat pig.animal`

Outputs content of pig.animal, which is bacon
(Aside: No, the output is oink, not bacon. Sadly.)

j. Did all of the cat commands succeed? What does this tell you about the meaning/purpose of file extensions (.txt, .pdf, .docx, etc.)?

File extensions are there to help clarify what file format a file is. You can name them whatever you want, but just because you can doesn't mean you should.

k. `cd ..`

Return to parent directory.

l. `rmdir foobar`

You should receive an error message:
`rmdir: failed to remove 'foobar': Directory not empty`

m. **(Be careful you type this command exactly!)** `rm -rf foobar`

Note that the -r flag means recursive, and -f for force.

Shell Scripting

a. Remember to always start your bash scripts with the line "`#!/bin/bash`".
b. If you try to run a shell script but you get the error "Permission denied", make sure that you have given that file execution permissions by running:

```
chmod +x <file-name>
```

NB: Don't just give execution permissions to any file! Before you execute any script, always make sure that you know who wrote it and what it does!

6. Write a shell script named "hello.sh" that, when run, outputs "Hello, world!"
Answer:

```
#!/bin/bash
echo "Hello, world!"
```

7. Write a shell script named "first.sh" that outputs whatever its first argument is (empty if no arguments are given).
Answer:

```
#!/bin/bash
echo $1
```

8. Write a shell script named “bridgekeeper.sh” that, when run, asks the user for their name with “What is your name?”, “What is your quest?”, “What is your favorite text editor?”, and responds with “You are [*insert name here*], on a quest [*insert objective here*], who’s favorite text editor is Emacs!”, ending with a newline.
    a. But can you do this without using an escape character?

Answer:

```
#!/bin/bash

emacs="Emacs"

read -p "What is your name? " name
read -p "What is your quest? " quest
read -p "What is your favorite text editor? " emacs

echo You are $name, on a quest $quest, who\’s favorite text
editor is $emacs!
```

Answer:

```
#!/bin/bash

emacs="Emacs"

read -p "What is your name? " name
read -p "What is your quest? " quest
read -p "What is your favorite text editor? " emacs

echo "You are $name, on a quest $quest, who’s favorite text
editor is $emacs!"
```

9. The following shell script prints out the number of visible files in the current directory:

count-files.sh:

```
#!/bin/bash

FILES=`ls`
COUNT=0
for FILE in $FILES
do
```

```
  # echo "Found file: $FILE"
  COUNT=$((COUNT+1))
done

echo "There are $COUNT visible files in this directory."
```

Can you use the above script to write a new one named “list-files.sh” that behaves exactly as the `ls` command (invoked with no arguments), except it prints out files separated by spaces instead of tabs, and terminates its output with a newline?

Hint: By default, `echo` always adds a newline character to the end of its output. You’ll need to figure out how to suppress this functionality.

Answer:

```
#!/bin/bash

FILES=`ls`
for FILE in $FILES
do
  echo -n "$FILE "
done

echo
```