Lab 09 - Exploring the Unix/Linux command line

This lab looks long, but it the actual questions don't take up a lot of space (or time!). The main goal is to get you to practice on the Unix command line. Please read the whole thing and follow every single instruction. You'll need the knowledge for next week!

Prerequisites: Having created your virtual Linux environment. See the instruction sheet.

0. Start your environment

Start VirtualBox, launch your virtual Linux machine and log into your account.

(Take a moment to explore your Graphical User Interface (GUI). For example, launch a Firefox browser and visit a website. More programs can be accessed with the nine-dot square icon in the lower left. Launch the Sudoku or Mines application, play a quick game, and close it again. The Ubuntu Software application allows you to install a whole lot of free software. Go ahead! But be careful, we didn't give our virtual Linux computer a lot of space.)

Find the Terminal application and open it. (Tip: You can right-click on the Terminal icon and select "add to favorites". Then it will appear in your applications shortcut panel on the left.)

In your Linux GUI, each window has three icons in the top-right, like on Windows: for minimizing, maximizing and closing the window. Maximize the terminal application so that your shell window occupies the maximum space. Then using either File>New Tab or SHIFT-CTRL-T add another tab (or two). You can have as many shell windows in parallel as you want!

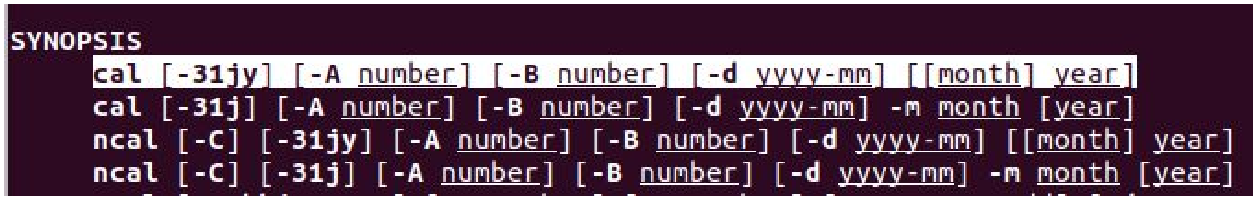
How you should hand in the solutions to this lab: From within your virtual machine, launch either the application "Text Editor" using the GUI icon, or type "`gedit &`" from a terminal window. This launches a nice, simple text editor. (`gedit` is the name of the program, and the & is to make it so that it doesn't block the command line while it is running. Don't worry too much about this point!) You can just type plain text, or if you like, use the Markdown codes like in GitHub .md files, see https://guides.github.com/features/mastering-markdown/ . Save the file as BtM2018_Lab09_yourname.txt (or .md) on the Desktop (inside the virtual machine!). Create the file now, before you even get started, and at the end use Firefox to log into your email account to mail it to Andrew (or, at first, yourself).

1. Explore simple commands

We'll use the cal command as an example for how to explore a simple, useful command. You can just type "`cal`" at the command prompt. Can you guess what it does?

For more information, type "`man cal`" and read the beginning of the manual page.

Under Synopsis, you'll find the first example of how to use cal:



```
SYNOPSIS
     cal [-31jy] [-A number] [-B number] [-d yyyy-mm] [[month] year]
     cal [-31j] [-A number] [-B number] [-d yyyy-mm] -m month [year]
     ncal [-C] [-31jy] [-A number] [-B number] [-d yyyy-mm] [[month] year]
     ncal [-C] [-31j] [-A number] [-B number] [-d yyyy-mm] -m month [year]
```

So let's explore these options! They can be used individually or in combination Type in one by one, and figure out what the output means. The man page can help. (Tip: Keep the man page open in one tab of the Terminal, and type your commands in another tab. To quit the man page, type q.)

```
$ cal -3
$ cal -1
$ cal -j
$ cal -y
$ cal -3j
$ cal -A 2
$ cal -A 2 -B 3
$ cal 7
$ cal -3 7
$ cal -d 2017-2
$ cal 5 2017
$ cal 2016
```

If you don't understand every single one of these, ask for help! First check `man cal`, then ask your fellow students, then ask your instructors.

Question section:

1.1 What weekday was Halloween (Oct 31) in 2015?
1.2 Type "`cal -3 9 1752`" . Is there anything that looks odd? Describe.
1.3 A similarly useful command is `date`. Type it and provide the output.

(If you're interested in the "why" of your observation in 2.2, you can read up on the history in http://mentalfloss.com/article/51370/why-our-calendars-skipped-11-days-1752).

2. Practice moving around the file system tree

Carry out the following actions one by one. You'll need the slides from the lecture, or any of the two reference cards.

- Go back to your home directory using "`cd`" (or "`cd ~`" or "`cd $HOME`" -- they all do the same thing).
- Create a directory called `dir1` using `mkdir ("mkdir dir1")`.
- Create a directory called `dir2` using `mkdir`.
- Use `cd` to change into `dir1 (cd dir1)` and back up to your home directory (`cd ..`) . do this a few times.
- Change directly to the directory `/usr` using `cd` . **NOTE**: "`cd /usr`" is a change to an *absolute* directory path: the directory called "`usr`" under the root directory ("/"). "`cd dir1`" is a change to a *relative* directory path: the directory "dir1" under whatever directory you happen to be in. Try "`cd dir1`" after "`cd /usr`" -- you will get an error message!
- Change back to your home directory.
- List all the files and directories in it, using ls. (Try also `ls -a, ls -l, ls -la`.)
- Change into `dir1`. Then type "`cd ../dir2`". Use pwd to find out what directory you're in.

Question section:

2.1 At the end of this sequence, what is the (full) path of the directory you are in?
2.2 In the directory /usr, how many subdirectories are contained in it? How many files (directly in it, not inside subdirectories)?

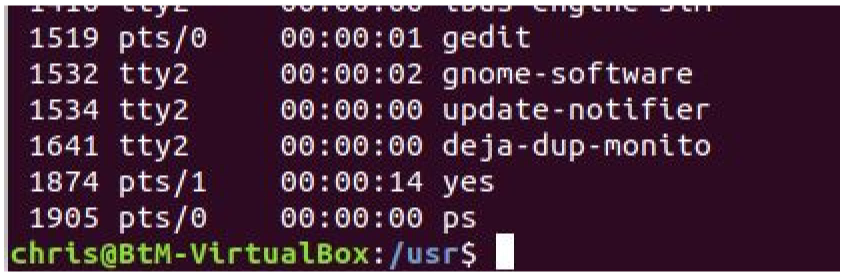
3. Get information about your environment

In this section you will try out a few more commands we have not yet seen in class. Use the FOSSwire Unix/Linux command reference page, and/or your (updated) slides for reference.

- `uname` provides information about the operating system. Try "man uname" and then call the command using the options -s -n -m -r (one by one!) and the most commonly used option, -a ("all") .
- The commands `hostname`, `whoami`, `uptime` are quite self-explanatory.
- `du` ("disk usage") shows how much space is consumed by all subdirectories under a directory. We usually use it with two options: `-h` "human readable" for a more friendly display of the directory size, and a long option to restrict how "deep" the command digs down into a directory: `--max-depth`. For example in your home directory, try `du`, `du -h`, `du --max-depth 1` and `du -h --max-depth 1`.
- `top` provides a self-updating display of all currently running processes and the resources they consume. There's a lot of information there - just try to understand some basics, such as user, command name, CPU % and MEM(ory) % . (For your information, Gnome is

the name of the graphical user interface that comes with Ubuntu Linux. Graphical user interfaces tend to use a lot of memory!)

- `ps` lists all active processes and has a *lot* of options. Try `ps -a.` Find the process ID in the first column and the command in the last column. On my computer, the output looks somewhat like this:



```
 1519 pts/0    00:00:01 gedit
 1532 tty2     00:00:02 gnome-software
 1534 tty2     00:00:00 update-notifier
 1641 tty2     00:00:00 deja-dup-monito
 1874 pts/1    00:00:14 yes
 1905 pts/0    00:00:00 ps
chris@BtM-VirtualBox:/usr$ 
```

  This is after I have started, in a different tab, an infinite process using the `yes` command, which indefinitely outputs a string (try "`yes ARRRRGH!`", which you will have to stop typing CTRL-C.)

- You can use the output of `ps -a` to kill processes. For example, I could kill the `yes` process in different ways:
  - `killall yes` ← the argument is the command name
  - `kill 1905` ← the argument is the process ID
  - `kill -9 1905` ← kill has an option: the killing strength. 9 is strongest.

  (Some processes are hard to kill and need `kill -9`. You may have noticed there are a lot of jokes in Unix land...)

Question section:

3.1 Using `top`, find out which process is currently using the most memory. Provide the name of command and user who owns it.
3.2 Provide the output of `uname -a`, `hostname`, `whoami`, and `uptime`.
3.3 For this question, you'll need two Terminal tabs. In one tab, type "yes ARGH!" (the yes command creates an infinite loop by indefinitely repeating some text to the command line). In the second tab, type `ps -a.` Find the process ID that goes with the `yes` command. Then kill that process. Provide the exact command, and what is written in the second tab after the ARGH! once the process is killed.

4. Getting information about files, and more practice with wildcards, redirection and pipes

For this lab, we will simply use the files we already have and can easily create. One file that all of you have in your home directory is called `.bashrc` . It is a hidden file, starting with a dot. It contains configuration options for your bash environment. Let's make a copy of it, and name it `mybashrc`: `cp .bashrc mybashrc`.

- To output the contents of `mybashrc` to the command line, use `cat mybashrc`
- To append the contents of `.bashrc` to the end of `mybashrc`, use a redirection: `cat .bashrc >> mybashrc`
- To read the contents of `mybashrc` page by page, use `more mybashrc,` or `less mybashrc .` Type `q` to get out of the pager. (`more` used to be an older version of the programm, and `less` is the newer version, but on your Linux system the two actually do the same thing.)
- To count the number of words in mybashrc, you pipe the contents of the file through the command `wc -w`. For the number of lines, it is wc -l: `cat mybashrc | wc -l`.
- **Do this only after you've answered the questions below:** 1. Practice renaming a file, using mv ("move"): `mv mybashrc myoldbashrc`. 2. Check with `ls` that `mybashrc` isn't there any more. Instead you have `myoldbashrc`. 3. Delete your `myoldbashrc` test file using the `rm` command. **Be careful not to delete `.bashrc`!**

Question section:

4.1 Compare the number of lines in `mybashrc` with the number of lines in `.bashrc` (run `cat filename | wc -l` for each of the files.) Why is one a multiple of the other?
4.2 What is the last line in mybashrc?
4.3 How would you append a line that says "This is the end of the file"? (Tip: remember the `echo` command, and redirection.)