

Linux and Virtual Machine

Student ID: _____

Name: _____

Assigned: Wednesday Jan. 9

Due: Wednesday Jan. 16 at 11:59pm.

Late submissions will be accepted for only a few hours after the deadline. Submissions between 12am and 1am (Jan. 18th) : 3% penalty, between 1-2am: 9%, 2-3am: 18%, and 3-4am: 30%. No submissions accepted after 4am.

Notes:

This assignment based on individual work. No collaboration is allowed (Discussing the technical details of each problem with other students before submitting your work, copying from students of current or previous semesters is not permitted in our lab and HW assignments). You may post your questions on the discussion forums and/or use the office hours. We may pick some students in random to demonstrate their design and simulations. Please refer to the syllabus and the first lecture regarding the USC and our policies, including the penalties for any violation. If you have any doubts about what is allowed or prohibited in this course, please contact the instructor.

What You Will Practice

In this lab you will learn the basic commands and navigation of Linux. You will write a very simple python code and get to know the procedure of submission programming assignments and labs.

Part 1: Linux

1. Software to Install on your PC

Start by following the virtual machine installation plan available on Blackboard. Budget a couple of hours for this as early as you can. In case of problems, visit the office hours of course staff.

Note: In Virtual Machine terminology we refer to the 'host' OS which is the actual OS your PC is running (OS X or Windows) and the 'guest' OS which is the OS running virtually (i.e. Ubuntu).

2. Getting Started with Linux

Background:

UNIX was developed by AT&T Bell Labs in 1969. The operating system was first meant to be used internally in Bell Systems (back then parent company of Bell Labs and AT&T), however, in the late 1970s, they licensed the software. This led to a variety of both academic and commercial Unix variants from vendors including UC Berkeley (BSD), Microsoft (Xenix), IBM (AIX) and Sun Microsystems (Solaris). In 2000, Apple released Darwin, also a Unix system, which became the core of the Mac OS X operating system, which was later renamed macOS. Currently, the Unix version with the largest installed base is Apple's macOS.

In 1987, MINIX was created as a minimal Unix-like operating systems targeted at students and others who wanted to learn the operating systems principles. In 1991, while attending the University of Helsinki, Torvalds became curious about operating systems. He began to work on his own operating system kernel, which eventually became the Linux kernel.

File System and Navigation Commands

It is important to understand how directories are arranged in Unix/Linux. Logically, Unix files are organized in a tree-like structure (just like in Windows). '/' is the root directory (much like C: is for Windows). Underneath the root are other directories/folders with a sample structure shown below.

The /home directory contains the user files for all accounts on the system. Other top-level directories include applications, libraries, drivers, et cetera. The name of the account you will use is student and so your files will be located in the student subdirectory of /home, which can also be referred to as /home/student or using the shortcut name ~ (tilde sign).

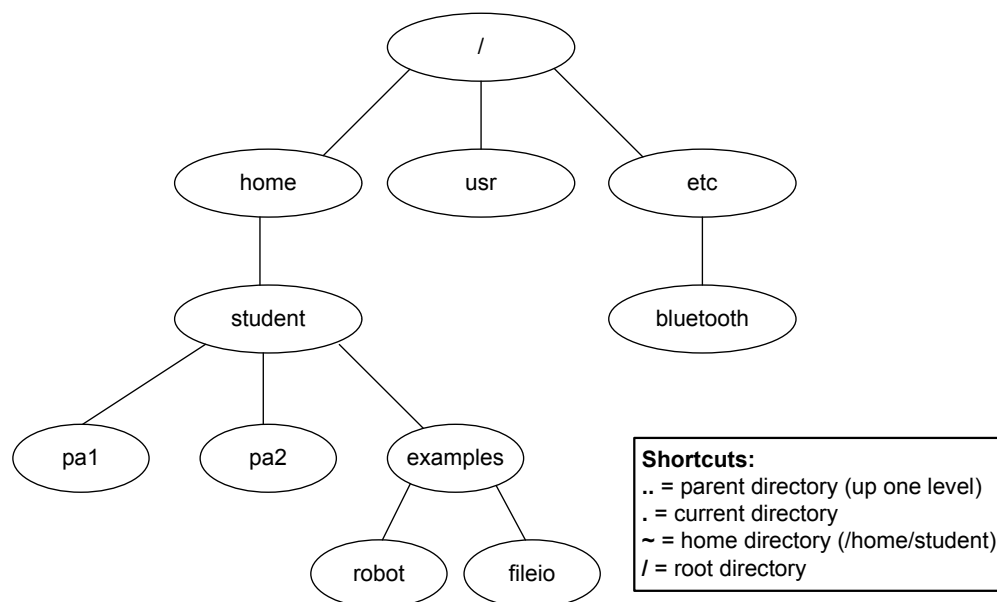


Figure 1- Example Unix/Linux File System Structure




After you log into the system, click the icon shown at the left. This opens the Terminal, a text-based window for interacting with the system. It is also sometimes called the “command line.” Unlike Windows or OS X, you will primarily interact with the system by typing in commands. In fact, those systems have terminals too (called Terminal in OS X and Command Prompt in Windows). The commands you will learn are very similar to what you can do in OS X, and fairly similar to what you can do in Windows.

Here is a crash course to introduce three of the most important commands for text-based file system navigation:

- `cd`: change directory. At every moment, your Terminal window is “visiting” a particular folder called its current “working directory”. The `cd` command is used to navigate from on working directory to another.
- `pwd`: report your present working directory (where you are now)
- `ls`: list the files directly inside of your present working directory

Enter these commands **exactly in the order shown below**. They’ll show the variety of ways in which these commands can be used. You have to press Enter/Return after each command.

Type this:	Effect:
<code>cd /</code>	Go to the root directory of the file system
<code>pwd</code>	Prints <code>/</code> (present working directory is filesystem root)
<code>ls</code>	Prints the files and folders directly inside of <code>/</code> : <code>bin boot cdrom dev ...</code>
<code>cd home</code>	Enter the home subdirectory of <code>/</code> . Note, the terminal shows <code>student@course-vm:/home \$</code> which is just a reminder of who and where you are.
<code>ls</code>	Prints the one folder directly inside of <code>/home</code> : <code>student</code>
<code>cd student</code>	Enter the student subdirectory of <code>/home</code> .
<code>pwd</code>	Prints <code>/home/student</code> (you navigated to your home dir)
<code>ls</code>	Prints contents of your home dir (including Desktop)
<code>ls -al</code>	Prints hidden files and folders, and all file information. Note that it includes <code>..</code> which is a parent shortcut:
<code>cd ..</code>	Move to parent of current directory (back to <code>/home</code>) 
<code>pwd</code>	Prints <code>/home</code> (you just navigated here)

<code>ls /usr</code>	Directly show contents of /usr
<code>ls /usr/gam</code> and then press Tab instead of Enter	The Terminal autocompletes gam to games/ Autocompletion is extremely useful! Press Enter to list contents of /usr/games
<code>/usr/games/sol</code>	Run the sol program in /usr/games Close it whenever you're ready to move on...
<code>cd ~</code>	Return to your home directory
<code>pwd</code>	Prints /home/student

In general, you can use:

```
cd <directoryname>
```

to enter a subdirectory of the current one,

```
cd /<directoryname>/<directoryname>/
```

to enter an absolute location, and you can always use ~ as a shortcut to your home directory and .. as a shortcut to the parent or the current directory.

Many commands such as `ls` can either be typed plain or with “command-line arguments” like `ls -al`. Later in the course we’ll learn about how to utilize these arguments in our own programs.

3. Editing and Organizing Files

Unix/Linux systems permit many text editors, though for beginners we recommend `gedit` which is simple and powerful (multiple tabs, search-and-replace, etc.).

It’s up to you how you want to organize your files on your virtual machine, but we recommend using a folder called `ee355` inside of your home directory (or inside of Dropbox). (Later in the semester you may want to organize your files into subfolders for each assignment.) Here’s how to create a subdirectory and text file:

<code>cd ~</code>	Go to your home directory. (If you’re using Dropbox for backup, use <code>cd ~/Dropbox</code> instead.)
<code>mkdir ee355</code>	Create ee355 subdirectory

<code>cd ee355</code>	Enter it
<code>gedit hello.txt</code>	Start editing a new text file. Type something in and save. (To save, use the menu at the top of the screen OR the buttons just above the editing area.) While gedit is still open, try typing <code>ls</code> at the Terminal. It's frozen... we'll explain shortly. After, quit gedit and try typing <code>ls</code> at the Terminal. You should see <code>hello.txt</code> listed.
<code>more hello.txt</code>	Show what's inside of <code>hello.txt</code> (what you typed in).
<code>gedit hello.txt &</code>	Open <code>hello.txt</code> for editing AND (using <code>&</code>) let Terminal continue taking input. Edit and save. Don't quit gedit.
<code>more hello.txt</code>	See the changes you made. Afterwards, quit gedit.
<code>rm hello.txt</code>	Delete (remove) the file <code>hello.txt</code> .
<code>ls</code>	Now <code>hello.txt</code> is gone. (You'll still see the backup file <code>hello.txt~</code> that gedit creates automatically.)

Linux/Unix has many other useful commands:

- `man` (manual) which is the "help" command. Try running `man ls`
- `clear` which clears your terminal screen
- `rmdir` which removes a directory
- `mv` which renames or moves a file (an example is given further below)
- `cp` which copies a file (using the same syntax as `mv`)
- `ps` which lists all the processes currently running. Try opening `/usr/games/sol &` and then look at the output of `ps`.
- `kill` which terminates a program. Try killing the program number that `ps` listed next to `sol` and see what happens.
- `grep` which searches for text. What is the output of this command?
`grep NAME /etc/os-release`
- Download `hi.txt` from online: `wget ftp://bits.usc.edu/hi.txt`
What's inside of it? (use `more`)

Part 2: Let's Python, Test and Submit!

In this part we will practice writing a simple python code. Even though we are in the beginning of our class, you will see that how simple it is to search our coding questions online. At the end, we provide you with the information of how you should prepare your code for submission, and how we check it with automatic scripts.

1. Hello World:

Let's create an 'ee355' directory in your home directory. Note: the '\$' represents the command prompt in the commands below. Type what is shown after the '\$'.

```
$ cd ~  
$ mkdir ee355
```

Navigate into the ee355 folder using the 'cd' command:

```
$ cd ee355
```

While the command prompt on the screen should show you what directory you are in, you also see your current directory path by typing the following:

```
$ pwd
```

'pwd' stands for 'present working directory' and will print out your current location. Rather than typing that all the time it may be beneficial to see what directory you are in displayed along with every command prompt. If you want to return to your home directory, we can go "up" one level to the **parent** directory using '..' [Note the **current** directory can be referenced using a single dot, '.']. Execute the commands.

Let us write our first python program. We will need to create and edit a text file that contains our source code.

```
$ gedit hello.py &
```

Recall that previously when gedit was running the command prompt froze until we exited gedit. This time when we start gedit we added the '&' at the end. This tells the shell/command prompt to 'fork' off the new application and continue accepting commands. In the gedit window type in the following program, verbatim.

```
# EE355  
# Lab1  
# Author: Tommy Trojan  
# Description: Yeay! This is my first python code (at least in EE355)  
  
print Hello World!  
print 12*13  
print 12.12*13.13  
print 12.121*13.131
```

Save the file (hello.py) and go back to the command window leaving gedit open so we can fix any errors if they exist. At the command prompt type:

```
$ python hello.py
```

Notice an error is output from the interpreter there exist an invalid syntax. That is because strings in python should be within quotation marks. Fix the code and run it again.

2. Getting input from user and printing values

Create a new python file and name it: "EE355_Lab1_<STUDENT-ID>.py"¹. Copy and paste the structure code below in your file:

```
# EE355
# Author: Tommy Trojan
# Lab1
# Description:
# Getting input from user
# simple arithmetic calculations
# changing the default setting for printing numbers

variable = input("Please type a number: ")
print "Input number: "
print variable
print variable*variable
# Complete below
# Print the cube of the variable with three decimal digits
```

When you run your code, a prompt will pop up and asks you to enter a number, it prints the variable, and also its square value. Try your code with different inputs, from different data types such as real numbers, integers, or even a string.

3. Printing with precision

Enter 12.345 as your number, and you can see the result has many digits after decimal point. In our application, we don't require such precision and it reduces the readability of our output. Try your code again, this time with an integer value like 2, and you will see the result will be an integer (4). The problem with this output is that the reader is not sure if the result is rounded to an integer or is precisely 4. In other words, there is more information in printing "4.000" than "4".

In this part we would like to print the cube of our input with 3 fraction digits precision. We did not learn how to do this in our course, but we can see how easy it is to check it online. Search this query in Google: "set precision of python print floating point", click on your first suggestion². Read the question, it may not be exactly your question, but the first answer (selected based Stack Overflow algorithms based on different features such as number of users' feedbacks) clearly explains how to set precision for printing floating numbers. Now apply what you have learned here and set the print precision as 3 digits.

4. Test your code before submission

Test your code with different values and check your results manually. You can **redirect** the output stream of your code to a file, instead of being printed on the console/terminal ">" character as

¹ Please replace <STUDENT-ID> with your own 10 digit USC-ID, the name of the file would be something like: EE355_Lab1_1234567890.py

² It should be [this](#) Stack Overflow page.

below. All the prompts will also be sent to the output file, but inputs are still coming from your terminal. Follow the steps below to understand what is really happening:

```
$ python EE355_Lab1_1234567890.py > output.txt
12.345
$ cat output.txt
```

The content of the file is shown below:

```
Please type a number: Input number:
12.345
Square:
152.399025
Cube:
1881.366
```

We have given this file to you as `output_golden.txt`. Use the command `diff` in your terminal to see there are any **differences** between these two text files. There should be no single difference between these two files.

```
$ diff output.txt output_golden.txt
```

5. Testing your file with script:

We use shell scripts³ to automatically check your assignments. We will give you a simple version of our script (here is called `script.sh`) and golden files (here is called `output_script_golden.txt`) for your convenience, and to avoid any points reduction due simple mistakes of spaces, new lines, upper/lower case letters, etc. Please run this command in your terminal:

```
$ sh script.sh
```

In this script we are checking your code with three different input values: `12.345`, `1.2`, and `-1`. The script will prompt you if the test has been failed or passed. In case it is failed, try to debug. We will use a similar script, but with more test cases to grade your work.

Note: we will not give you incorrect input values such strings.

Note: There should be only one single file names as `EE599_Lab1_*` in your directory when running the script.

6. Submit

Upload only your `EE355_Lab1_<STUDENT-ID>.py` under assignments in Blackboard.

³ You will learn about shell, and shell scripts very soon! No need to know about them for now!