Computer Science 507
Software Engineering
The College of Saint Rose
Spring 2015

Lab 1: Unix Introduction/Refresher and Other Setup

Due: 6:00 PM, Monday, January 26, 2015

In this first lab, we will set up some accounts and get you used to using them. This includes a chance to learn or refresh your knowledge of some basic Unix commands, and to write and run some simple C and Java programs in a Unix environment.

You may work alone or in groups of size 2 or 3 on this lab. Only one submission per group is needed.

You may ask your instructor and classmates for help as you complete this lab, but the work you submit must ultimately be your own (or that of your group). If you are completely unfamiliar with Unix, don't hesitate to ask questions! On the other hand, if you have some experience, don't hesitate to help a classmate! None of these tasks is intended to be hard, but if you don't have much Unix experience (and it is reasonable if you don't), they could be.

Getting Set Up

Create a document where you will record your answers to the lecture assignment and lab questions. If you use plain text, call it "`lab1.txt`". If it's a Word document, you can call it whatever you'd like, but when you submit, be sure you convert it to a PDF document "`lab1.pdf`" before you submit it.

Also, read over the description of the types of items you will encounter in our labs on the course home page.

Lecture Assignment Questions

We will usually discuss these questions at the start of class on the lab due date, so no credit can be earned for late submissions of lecture assignment questions.

Before attempting these problems, please complete the reading assignment listed on this week's lecture page.

? LA Question 1:

Sommerville Exercise 1.1, p. 25. (2 points)

? LA Question 2:

Sommerville Exercise 1.2, p. 25. Also give examples of a generic software product and a custom software product. (2 points)

LA Question 3:
Sommerville Exercise 1.7, p. 25. (2 points)

LA Question 4:
Sommerville Exercise 2.1, p. 54. (2 points)

LA Question 5:
Sommerville Exercise 2.2, p. 54. (2 points)

LA Question 6:
Sommerville Exercise 2.7, p. 55. (2 points)

LA Question 7:
Sommerville Exercise 3.2, p. 78. (2 points)

LA Question 8:
Sommerville Exercise 3.3, p. 78. (2 points)

LA Question 9:
Sommerville Exercise 3.6, p. 79. (2 points)

LA Question 10:
Sommerville Exercise 4.2, p. 116. (2 points)

LA Question 11:
Sommerville Exercise 4.7, p. 116. (4 points)

LA Question 12:
The Brooks paper was written over 25 years ago. Discuss which parts of the article you believe are still relevant today, and what in the article has been shown not to be true over the years. Include your thoughts on each of the items listed in the "Hopes for the Silver" section. (6 points)

SubmissionBox Account

We will use a system called SubmissionBox, developed by previous students of this course, for most assignment submissions this semester. If you have not had a class with me before that used SubmissionBox, you will need to have an account set up for you. Even if you have, you will need to be added as a student to this course. Please come to the teaching station to get your SubmissionBox

account set up.

Once your SubmissionBox account has been set up (or updated to include you as a student in this course, if you already had an account), create a document with a brief description of your background. Include the same information as in the "student introductions and backgrounds" item from class. Then log into SubmissionBox, and submit this document under assignment "SBTest". Verify that you received an email confirmation for this submission. You will earn 1 point for making this submission.

Blogging Site Account

We will be using a Wordpress blog site to discuss the speakers who will be presenting to the class throughout the semester. Please visit http://www.strosecs.com/cs507s15speakers/ and register for an account that you can use to comment on our speakers. For now, complete the registration process and add a comment on the post "This site will contain..." to make sure everything is working.

Unix Account

We will be using a virtual Linux server called `mogul.strose.edu` at times during the semester (and definitely today). If you already have an account on mogul, please make sure you can still log in. If you don't (or aren't sure), please come to the teaching station to have an account created for you.

Log into `mogul.strose.edu`. From PuTTY or similar Windows secure shell clients, just fill in the information on the connection dialog using `mogul.strose.edu` for the host name and your username for the username.

If using ssh from a Terminal window on the Mac, and your username on `mogul.strose.edu` is `jcool`, you would issue the command

```
ssh mogul.strose.edu -l jcool
```

at the terminal prompt. Log in with your `mogul.strose.edu` password. You should be presented with a prompt that looks something like:

```
[jcool@mogul ~]$
```

and mogul is now ready to accept your commands. More on those below.

Unix Practice

GUIs are nice, but they can be slow to navigate and too restrictive for some purposes. You can often work much more efficiently by working in a Unix environment and interacting with the system by typing commands at the Unix *shell*, or *command line*. When you log in, you will be presented with a prompt. This is your direct interface to issue commands to the operating system. When you type

a command here, the shell will execute the command on your behalf, print out any results, then reissue the prompt.

Of course, the command line is useless if you don't know what commands it understands. You will learn about several important commands in this lab, but we will only scratch the surface. One of the most important is `man` – the Unix manual. Every Unix command has a manual page, including `man`. To see the manual page about `man`, type the command:

```
man man
```

You will be presented with a manual entry about the Unix manual, one page at a time. You can advance to the next page by pressing the space bar, and can quit out of `man` to return to your command prompt by typing '`q`'.

The Emacs Editor

Emacs (`emacs` from the Unix command line) is a powerful text editor, which is very good for programming in a language like C and for general plain-text editing. You will need to become familiar with it.

To try it out, you will use it to create your `lab1unix.txt` file that will contain your answers to this week's lab questions. For now, you are to create this file in your home directory on mogul.

You already should have one session connected to mogul from above. Now open a second PuTTY or Terminal window and log into `mogul.strose.edu` on that one as well.

In one of the windows, launch `emacs` on the file `lab1unix.txt`.

```
emacs lab1unix.txt
```

Emacs should start up, and present you with a text-based menu across the top (which we will purposely ignore), a large area where you can edit the file, and two lines of status information across the bottom.

Type your name(s) and "Lab 1 Questions" in the Emacs window that is editing the file `lab1-unix.txt`.

In the other window, launch another `emacs` session where you can type some text and then identify the function of and experiment with the Emacs commands below. Note that `C-` before a key means hold down `Ctrl` and hit that key. `M-` indicates the "Meta" key, which on most systems is `Esc`. To issue a Meta command, hit the `Esc` key, release it, then hit the key(s) for the command you wish to issue. Use the keystrokes rather than the menus. It will save you time in the long run! Note: for some of these commands, a very small buffer (that is, the contents of the file you are editing) will not allow you to see what they do. So create a file with several screens full of text before you go too far.

```
C-x C-s   C-x C-c   C-x C-f   C-x C-w   C-g   C-a   C-e
C-d       C-_       C-v       M-v       C-s   C-r   M-%
C-k       C-y       C-x u
```

Question 1:
Complete your Emacs command descriptions in `lab1unix.txt` (4 points).

Directory Structure

It is always important, but especially so when working with the Unix command line, to know where the files in various directories (often called "folders" on Macintosh and Windows systems because of how they are visually represented in GUIs) you might be using are actually stored, and where and how those are accessible. More and more computer users are getting into the habit of placing all files in a "My Documents" folder or in cloud storage where a search can be used to find files. But we're computer scientists and we can do better. We will keep our files in an appropriate directory hierarchy. And we need to be aware where the files "live" – are they on the remote server, in the cloud, or on our own hard drive or flash drive?

On `mogul.strose.edu`, we find a standard Unix style environment. Each user has a *home directory* where only that user has permission to read and write files. Your home directory is the initial *current directory* or *working directory* when you first log in.

The working directory is where a program will look for files unless instructed to do otherwise. You'll hear Unix users asking a question like "What directory are you in?" and the answer to this is your working directory.

The command `pwd` will instruct the shell to print your working directory.

Question 2:
What is your home directory on `mogul.strose.edu`? (hint: use `pwd`) ($\frac{1}{2}$ point)

Note: the lab questions for this week are worth 1 point each unless otherwise specified.

You can also list the contents of your working directory with the command `ls`.

Question 3:
What output do you see when you issue the `ls` command on `mogul.strose.edu`? ($\frac{1}{2}$ point)

Other important operations to navigate and modify the directory structure are changing your working directory (`cd`), creating a new directory (`mkdir`), and removing a directory (`rmdir`).

Create a directory in your account for your work for this course (`cs507` might be a good name), and a directory within that directory for this assignment (`lab1` might be a good name). **Note: if working in a group, all group members should complete these steps, but only one person's information needs to be included in your lab question submissions.**

Question 4:
Change your working directory to the one you just created and issue the `pwd` command. What does this show as your working directory?

In your shell window and in your home directory (note: you can always reset your working directory to be your home directory by issuing the command `cd` with no parameters), issue this command:

```
uname -a > linux.txt
```

This will execute the command `uname -a`, which prints a variety of information about the system you are on, and "redirects" the output, which would normally be printed in your terminal window, to the file `linux.txt`.

Output Capture:
`linux.txt` for 1 point(s)

Look at the contents of the file `linux.txt` with the command:

```
cat linux.txt
```

Question 5:
What do you think the information in `linux.txt` means?

Unix Commands

Identify the function of and experiment with these Unix commands (a few of which you have already used):

```
ls    cd      cp      mv     rm     mkdir   pwd
man   chmod   cat     more   grep   head    tail
ln    find    rmdir   wc     diff   scp     touch
```

Question 6:
Give a one sentence description of each command. (4 points)

Using appropriate commands from the above list, move the `linux.txt` file you created in your home directory into the directory you created on mogul for your work for this assignment.

Show that this has worked by issuing the following command from inside of your course directory (but not inside the directory for this assignment):

```
ls -laR > ls.out
```

Then move the file `ls.out` into the directory for this assignment.

Output Capture:
`ls.out` for 2 point(s)

Using the Unix manual, your favorite search engine, or in discussion with your classmates, determine the answers to these questions:

Question 7:
How do you change your working directory to be "one level up" from the current working directory? (Give the command.)

Question 8:
Give two or three different ways to change your working directory to be your home directory. All likely involve the `cd` command, but will take different parameters.

Compiling and Running a Java Program in Unix

Our assumption in this course is that you are an expert programmer in some programming language, most likely Java. Even so, much of your experience as a Java programmer is likely from within an Integrated Development Environment (IDE). Here, we will see how to compile and run a Java application from the Unix command line.

See Example:
`/home/cs507/examples/hello`

For you to run this, you will want to copy the example to your own directory. Create a directory called `hello` under your directory for this lab and copy the C and Java files from the example into that directory.

Change to that directory and compile and run it:

```
javac Hello.java
java Hello
```

Now, edit the Java program so it prints out a second message, recompile and re-run, but this time redirecting your output to a file `java.out`.

Output Capture:
`java.out` for 1 point(s)

When we run a Java program from the Unix command line, any additional parameters we place after the program name when we launch the program are delivered to the program in the `args` array that is passed to the `main` method.

Finally, modify the Java program so it prints out a third message which assumes the first parameter passed is the name of the person running the program. So running

```
java Hello Joe
```

would output an additional line:

```
Thanks for running, Joe!
```

Redirect the output of this new program to `java2.out` when you put your own name in as the first command-line parameter.

Output Capture:
`java2.out` for 1 point(s)

Compiling and Running a C Program in Unix

C is a widely-used, general purpose language, well-suited to low-level systems programming and scientific computation. We will not study it in detail in this course, but it is worth a bit of our time to see how to develop, compile, and run a simple C program in a Unix environment.

We will initially study it assuming you have Java experience, focusing on the features that make C significantly different from Java. Fortunately, Java borrowed much of its syntax from C, so it is not difficult for a Java programmer to read most C programs.

C++ is a superset of C (that is, any valid C program is also a valid C++ program, just one that doesn't take advantage of the additional features of C++). C++ adds object-oriented feautures. In this course, we will look only at C, not C++.

A Very Simple C Program

We will begin by seeing how to compile and run a very simple C program (`hello.c`) in a Unix environment.

As part of the previous task, you copied a C program called `hello.c` into a directory in your account. Change to that directory and compile and run it:

```
gcc hello.c
./a.out
```

Things to note from this simple example:

- We run a program named `gcc`, which is a free C compiler.
- `gcc`, in its simplest form, can be used to compile a C program in a single file:

  ```
  gcc hello.c
  ```

In this case, we're asking `gcc` to compile a C program found in the file `hello.c`.

Since we didn't specify what to call the executable program produced, gcc produces a file `a.out`. The name is `a.out` for historical reasons.

- When we want to run a program located in our current directory in a Unix shell, we type its name.
  - For example, when we wanted to run `gcc`, we typed its name, and the Unix shell found a program on the system in a file named `gcc`.
  - How does it know where to find it? The shell searches for programs in a sequence of directories known as the *search path*. Try: `env`.
  - So if we want to run `a.out`, we should be able to type its name. But our current directory, always referred to in a Unix shell by "`.`", is not in the search path. We need to specify the "`.`" as part of the command to run:

    ```
    ./a.out
    ```

- Of course, we probably don't want to compile up a bunch of programs all named `a.out`, so we usually ask `gcc` to put its output in a file named as one of the parameters to `gcc`:

  ```
  gcc -o hello hello.c
  ```

  Here, the executable file produced is called `hello`.

- And in the program itself, let's make sure we understand everything:
  - At the top of the file, we have a big comment describing what the program does, who wrote it, and when. Your programs should have something similar in each C file.
  - We are going to use a C library function called `printf` to print a message to the screen. Before we can use this function, we need to tell the C compiler about it. For C library functions, the needed information is provided in *header files*, which usually end in `.h`. In this case, we need to include `stdio.h`. Why? See `man 3 printf`.
  - A C program starts its execution by calling the function `main`. Any command-line parameters are provided to `main` through the first two arguments to main, traditionally declared as `argc`, the number of command-line parameters (including the name of the program itself), and `argv`, an array of pointers to character strings, each of which represents one of the command-line parameters. In this case, we don't use them, but there they are.
  - Our call to `printf` results in the string passed as a parameter to be printed to the screen. The `\n` results in a new line.
  - Our `main` function returns an `int` value. A value of 0 returned from `main` generally indicates a successful execution, while a non-zero return indicates an error condition. So we return a 0.

- Notes for Java programmers:
  - Good news: much of the syntax of Java was borrowed from C, so a lot of things will look familiar.
  - There are no classes and methods, just *functions*, which can be called at any time. Any information a function needs to do its job must be provided by its parameters or exist in *global variables* – variable declared outside of every function and which are accessible from all functions.

Now, edit the C program so it prints out a second message, recompile and re-run, but this time redirecting your output to a file `c.out`.

Output Capture:
`c.out` for 1 point(s)

Practice Programs

Write your own Java and C programs named `Seq.java` and `seq.c` that clone some of the functionality of the Unix `seq` command. See the man page for `seq(1)` for details, but the program should take 1, 2, or 3 numeric parameters. To simplify, you may ignore all other command line options, and that all given parameters are valid integers.

Note that the command-line parameters in each case will come to you as strings, and will need to be converted to integers before they can be used. In each language there are multiple mechanisms that will allow you to do the conversion. One possibility in Java is to use the `Integer.parseInt` method, and in C to use the `strtol` function from the C standard library.

If your programs are presented with invalid command line parameters (*e.g.*, there aren't the right number of parameters, the parameters cannot be converted to integers, or they otherwise make no sense), your program should print an appropriate error message and exit.

Make sure your programs compile and run on mogul using `gcc` for the C program, and `javac` for Java.

Each program is worth 15 points.

Submitting

Before 6:00 PM, Monday, January 26, 2015, submit your lab for grading. Package up all required files into an appropriate archive format (`.tar.gz`, `.zip`, and `.7z` are acceptable) and upload a copy of the using Submission Box at `http://sb.teresco.org` under assignment "Lab1".

Note that you will need to transfer the files you created on `mogul.strose.edu` to the computer from which you will be making your submission, so they can be included in your archive. Windows users might want to consider "WinSCP" or "FileZilla", Mac users can use the builtin `scp` command at the Terminal, or use "FileZilla". There are many other options in both cases.

Grading

This assignment is worth 80 points, which are distributed as follows:

Feature	Value	Score
Lecture Assignment Questions	30	
"SBTest" submission in SubmissionBox	1	
Lab Question 1 (Emacs commands)	4	
Lab Question 2 (home directory)	$\frac{1}{2}$	
Lab Question 3 (`ls`)	$\frac{1}{2}$	
Lab Question 4 (`pwd` in new dir)	1	
Output Capture `linux.txt`	1	
Lab Question 5 (`linux.txt` contents)	1	
Lab Question 6 (Unix commands)	4	
Output Capture `ls.out`	2	
Lab Question 7 (`cd` up)	1	
Lab Question 8 (`cd` to home)	1	
Output Capture `java.out`	1	
Output Capture `java2.out`	1	
Output Capture `c.out`	1	
`Seq.java` Practice Program	15	
`seq.c` Practice Program	15	
Total	80	