Lab Exercises wk03 – Linux Command Line
Introduction to Using the Linux Command Line

Required Reading

Java Foundations – Chapter 2 – Data and Expressions
Chapter 3 – Sections 3.1 to 3.5
Chapter 4 – Section 4.8 – The `For` Statement

Instructions – PLEASE READ *(notice **bold** and <u>underlined</u> phrases)*

Lab Exercise has three parts:

A. Lab Demo – Watch Demo, reproduce it, show your TA
B. Exercises – To be started during the Lab, and completed by deadline
C. Submission – Submit specified exercise files by deadline

1. You are encouraged to work on these lab exercises in <u>teams of two</u>, however <u>each student must reproduce the demo and show the TA</u> their own running Java Project. Team members should alternate turns at the keyboard typing in the code. You will start working during the lab time, and it is likely that you will complete the demo but need to continue with the exercises later.
2. **Submission deadline: 2:30pm Friday Sept 20**. You should be able to complete the work if you have completed the required reading for the lab. Some topics discussed in this lab and needed for the exercises to be submitted may be seen later in this week's class! You may submit before the deadline, if you so prefer. You may resubmit again later without penalty provided you resubmit before the deadline.
3. The exercises are roughly presented in sequence so that you gradually advance with the material. It is highly recommended, for your own benefit, that you do ALL the exercises, and in the order provided. You will get lab participation points even if you do not finish all the exercises (but you do have to work on some of the exercises following your completion of the Demo in order to get full participation points).
4. For this lab, there are four files that you must submit for marking. If working in pairs you should submit only one version of the file and submit it for the group that you define. If you are working individually, you may still need to define a group with you as the only group member.
5. Keep a copy of everything you submit in your files.
6. Before you leave the CSIL labs, make sure that a TA looks at your work in order to receive your attendance and lab active participation marks.

7. Lab03 Intended learning outcomes

By the completion of the demo by the student, students should:

- Understand the basic use of the Linux Command Line interface
- Be able to use basic linux commands including man, ls, cd, cp, rm, pwd, and mkdir
- Be able to do simple editing via vim
- Be able to compile and run Java programs from the command line

Upon completion of the lab exercises, students should be able to write simple programs that use:

- String Class for manipulating strings
- Scanner Class for reading user input
- Math Class methods for computing Power and Square Root
- Random Class for generating random numbers in a specified range

A. Lab Demo – Presented by TAs, and repeated by Students

Students must observe demo presented by TA, then reproduce it.

Student must have the TA check off demo program completion by end of tutorial for full marks. Marks will be awarded for attendance but only partial marks for incomplete work at the discretion of the TA.

Even though students may work in teams, <u>each team member</u> must create a running project based on the demo for full marks.

The Linux Command Line

The first thing to say about Unix is that it's all about openness and choice[1]: more openness and choice than most people ever use. This is in stark opposition to the philosophy of Microsoft and Apple, which provide you with one proprietary environment and severely limited choice. If you find yourself cursing at Windows or OS X on a regular basis and asking "Why do I have to adapt to my computer. Why can't my computer adapt to me?" consider taking a serious look at Unix. You may like it better, and the parts you do not like, you can change.

One of the powerful capabilities of Unix is the command shell. It should be no surprise that there are choices here too. The two most common shells are bash (the default shell in CSIL) and tcsh. Don't like bash? You can change to `tcsh` with the `chsh` ('change shell') command.

The remainder of this demo will assume the CSIL Ubuntu defaults. The examples are from the account skristja, so you will not see the exact same output.

Unlike Windows, in Unix pretty much everything is case-sensitive. Check for this when a command does not work as you think it should.

When you are using the command line, file and directory names with spaces are awkward to use. Avoid them if possible.

The Single Most Useful Unix Command

The single most useful Unix command when you are learning to use the Unix command line shell is the `man` (output the 'manual') command. If you forget how to use it, try typing '`man man`':

```
skristja@asb9838n-a07:~$ man man
MAN(1)              Manual pager utils              MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man  [-C file] [-d] [-D] [--warnings[=warnings]] [-R
```

[1] Linux is a particular variant of Unix. Ubuntu is a particular distribution of Linux.

```
       encoding] [-L locale] [-m system[,...]] [-M path]
       [-S list] [-e extension] [-i|-I] [--regex|--wild-
       card] [--names-only] [-a] [-u] [--no-subpages] [-P
       pager] [-r prompt] [-7] [-E encoding] [--no-hyphen-
       ation] [--no-justification] [-p string] [-t]
       [-T[device]] [-H[browser]] [-X[dpi]] [-Z] [[section]
       page ...] ...
       man -k [apropos options] regexp ...
       man -K [-w|-W] [-S list] [-i|-I] [--regex] [section]
       term ...
       man -f [whatis options] page ...
       man -l [-C file] [-d] [-D] [--warnings[=warnings]]
       [-R encoding] [-L locale] [-P pager] [-r prompt]
       [-7] [-E encoding] [-p string] [-t] [-T[device]]
       [-H[browser]] [-X[dpi]] [-Z] file ...
       man -w|-W [-C file] [-d] [-D] page ...
       man -c [-C file] [-d] [-D] page ...
       man [-hV]

DESCRIPTION
       man is the system's manual pager. Each page argument
       given to man is normally the name of a program,
       utility or function. The manual page associated
       with each of these arguments is then found and dis-
       played. A section, if provided, will direct man to
       look only in that section of the manual. The
       default action is to search in all of the available
       sections, following a pre-defined order and to show
       only the first page found, even if page exists in
       several sections
```

... followed by a complete description of how to use the man command ...

This looks pretty intimidating, but you just have to know how to read it. Each line starting with `man` is one form of the command. Let's take one of the lines:

```
man -f [whatis options] page ...
```

There is a command line flag '-f', which will affect the way the man command behaves. Down in the description of the man command, you find

```
Main modes of operation
   -f, --whatis
   Equivalent to whatis. Display a short description from the manual
   Page, if available. See whatis(1) for details.
```

When you see text in square brackets, as '[whatis options]', the square brackets indicate an optional parameter. You can add options appropriate to the whatis command. (What might those be? Try 'man whatis'.) And then you must supply a page. Down in the description of the man command, you find

```
Each page argument given to man is normally the name of a program,
utility or function. The manual page associated with each of these
arguments is then found and displayed.
```

The `apropos` ('apropos of ') command (an alias for 'man -k') is useful when you're not sure what you're looking for. For example, you want to know how to list the contents of a directory, but don't

remember the command name:

```
skristja@asb9838n-a07:~$ apropos  -a list directory
chacl (1)            - change the access control list of a file o...
dir (1)              - list directory contents
File::Listing (3pm)  - parse directory listing
ls (1)               - list directory contents
ls (1posix)          - list directory contents
ntfsls (8)           - list directory contents on an NTFS filesystem
vdir (1)             - list directory contents
```

Typically, you will be interested in commands in section (1) of the manual. Now you can use the man command to find out more about these commands.

You will also want to know about the info command, and the Xubuntu help, the 'Help' entry towards the bottom of the Applications Menu[2].

Navigating the File System

As with all major operating systems, a Unix file system is a hierarchy of directories and files. For those of you coming from the Windows or Mac worlds, directories are exactly equivalent to folders. If you'd rather use a GUI, click on any of the default desktop icons ('Home', 'File System'), or the folder icon in the bottom (pop-up) panel. The GUI file browser is called thunar; you can start it from the command line by typing the command

```
skristja@asb9838n-a07:~$ thunar
```

Where Am I?

To make sure we all start in the same place, execute the `cd` command:

```
skristja@asb9838n-a07:~$ cd
```

The cd ('change directory') command, with no arguments, will place you in your home directory. (Here in CSIL, that's the local home directory for the workstation.)

The pwd ('print working directory') command will print the path to the current directory:

```
skristja@asb9838n-a07:~$ pwd
/home/skristja
```

Of course, you should see your user id, not `skristja`.

What's Here?

The ls ('list') command will list the files and subdirectories in the current directory (the set of files you see may not be the same as shown here):

```
skristja@asb9838n-a07:~$ ls
android-sdk                 Documents         Music     sfuhome
C:\nppdf32Log\debuglog.txt  Downloads         Pictures  Templates
Desktop                     examples.desktop  Public    Videos
```

Unix, like Windows and OS X, has hidden files and directories that hold configuration and state information used by the operating system and various programs. To see these, add '-a' to the command line:

[2] The menu at the top left of the screen, under the mouse head icon.

```
skristja@asb9838n-a07:~$ ls -a
.              Documents          .local         Templates
..             Downloads          .mozilla       Videos
android-sdk    examples.desktop   Music          .viminfo
.bash_logout   .gconf             Pictures       .Xauthority
.bashrc        .gnome2            .profile       .Xdefaults
.cache         .gnome2_private    Public         .xscreensaver
.config        .gstreamer-0.10    .pulse         .xsession-errors
.dbus          .gvfs              .pulse-cookie
Desktop        .ICEauthority      sfuhome
```

Yes, there are lots of them. For example, the `.config` directory is used by the Xfce window manager to store configuration information. The file `.bashrc` is executed by the bash shell as it starts up. This is the place to put your own custom command aliases or remove default aliases that you don't like.

How can I tell which entries are files, and which are directories? On many systems, `ls` will produce color-coded output, but if not, you can use the '`-F`' flag:

```
skristja@asb9838n-a07:~$ ls -F
android-sdk@  Downloads/        Pictures/  Templates/
Desktop/      examples.desktop  Public/    Videos/
Documents/    Music/            sfuhome/
```

Directories have a '/' added to the end. Executable files have an '*' tacked on the end. Plain old files have nothing added. Other characters are used to indicate other file types.

If you give a file or directory name, ls will restrict itself to just that file or directory. Suppose that I want some details for sfuhome. I can add the '-l' and '-d' flags to my ls command, with the directory name, and get:

```
skristja@asb9838n-a07:~$ ls -ld sfuhome
drwx--s--x 11 skristja users 0 Sep 14 13:03 sfuhome
```

What does this output mean? Here's a quick description of the fields that are interesting in an introduction:

- `drwx--s--x` are the file permissions. These permissions show that `sfuhome` is a directory (the leading 'd'); that I can read (browse), write (create), and execute (access) the directory (the next three characters, '`rwx`'); and that others can execute files in my `sfuhome` directory only if they know the file name (the characters '`--x`' mean that others can access the files but not browse them). The `chmod` ('change mode') command is used to manipulate file permissions. For optional reading on Linux permission bits, check out http://danielmiessler.com/study/unixlinux_permissions/
- `skristja` is the owner of the file.
- 0 is the size in bytes. (Recall that sfuhome is mounted from a file server in ITS; the mount itself occupies no space.)
- `Sept 14 13:03` is the time the directory was last modified.

Notice that many Unix commands use single-letter options, and you can combine those options with a single '`-`'. Try '`ls -alF`'.

Suppose you just want to see the files with extension `.txt`? Unix command shells support wildcard characters in file and directory names. The character '*' matches any string; '?' matches any single character.

```
skristja@asb9838n-a07:~$ ls *.txt
C:nppdf32Logdebuglog.txt
skristja@asb9838n-a07:~$ ls -d ?o*
Documents  Downloads
```

In the second `ls` command, I asked for a listing of any name that has 'o' as the second character. The directories Documents and Downloads matched.

Your ITS Home Directory

Your ITS home directory is mounted in the sfuhome directory.

How Do I Move Around?

To move to another place in the file system, use the `cd` ('change directory') command:

```
skristja@asb9838n-a07:~$ cd sfuhome
skristja@asb9838n-a07:~/sfuhome$ pwd
/home/skristja/sfuhome
```

Notice the change in the command prompt:

```
skristja@asb9838n-a07:~$
```

changed to

```
skristja@asb9838n-a07:~/sfuhome$
```

The default command prompt displays several pieces of information:

```
userid@hostname:current_directory$
```

Remember, in a Unix command shell the character '~' in a file name is the abbreviation for your home directory. You can, of course, change the prompt to anything you like. Consult the documentation for `bash`.

How Do I Create a Directory?

The `mkdir` ('make directory') command will create a new, empty directory:

```
skristja@asb9838n-a07:~/sfuhome$ mkdir Demo
skristja@asb9838n-a07:~/sfuhome$ ls -F
bin/      CMPT 150/  desktop.ini*  pub_html/
cmpt125/  Demo/      personal/     $RECYCLE.BIN/
```

What's in a brand new directory?

```
skristja@asb9838n-a07:~/sfuhome$ ls -aF Demo
./  ../
```

In unix file systems, every directory will have two entries:

- A single '`.`' (period) means 'the current directory'.
- Two periods, '`..`', means 'the parent directory'.

You can use these abbreviations on the command line:

```
skristja@asb9838n-a07:~/sfuhome$ cd Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ pwd
/home/skristja/sfuhome/Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ ls ..
```

```
bin      CMPT 150  desktop.ini  pub_html
cmpt125  Demo      personal     $RECYCLE.BIN
skristja@asb9838n-a07:~/sfuhome/Demo$ cd ..
skristja@asb9838n-a07:~/sfuhome$
```

How Do I Delete a File or Directory?

To experiment with deleting files and directories, we need a worthless file or two. One way to create an empty file is with the `touch` command:

```
skristja@asb9838n-a07:~/sfuhome$ cd Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ touch worthless
skristja@asb9838n-a07:~/sfuhome/Demo$ ls -l worthless
-rw-r--r-- 1 skristja users 0 Sep 14 16:45 worthless
```

Another way is to redirect some output into a file:

```
skristja@asb9838n-a07:~/sfuhome/Demo$  echo 'Hi, Mom!' > himom.txt
skristja@asb9838n-a07:~/sfuhome/Demo$ ls -l himom.txt
-rwxr--r-- 1 skristja users 9 Sep 14 16:48 himom.txt
```

The `echo` command simply echoes its argument. The '>' character redirects that output into the file `himom.txt`. Be sure to use single quotes around 'Hi, Mom!' The exclamation mark '`!`' has special meaning to the shell (beyond the scope of this introductory demo). The file `himom.txt` really isn't executable[3]. You can remove the execute permission with the command '`chmod a-x`':

```
skristja@asb9838n-a07:~/sfuhome/Demo$ chmod a-x himom.txt
skristja@asb9838n-a07:~/sfuhome/Demo$ ls -l himom.txt
-rw-r--r-- 1 skristja users 9 Sep 14 16:48 himom.txt
```

To delete a file, use the `rm` ('remove') command:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
himom.txt  worthless
skristja@asb9838n-a07:~/sfuhome/Demo$ rm worthless
rm: remove regular empty file `worthless'? y
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
himom.txt
```

Normally, the Unix command line does what you tell it, immediately, with no argument. The `rm` command above prompted for confirmation because the default CSIL configuration creates an alias:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ alias rm
alias rm='rm -i'
```

The '`-i`' (interactive) flag is the reason for the prompt. If you're tired of having your computer ask 'Are you sure?' every time you ask it to do something, simply remove the alias:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ unalias rm
skristja@asb9838n-a07:~/sfuhome/Demo$ touch worthless
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
himom.txt  worthless
skristja@asb9838n-a07:~/sfuhome/Demo$ rm worthless
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
```

[3] This is an artifact of sharing your ITS home directory between unix and Windows using a common file system standard called CIFS. CIFS has difificulty with unix file permissions. Technical staff are trying to work around this; at present the behaviour seems to be inconsistent from one workstation to the next.

```
himom.txt
```

If you like the opportunity for a second thought, it's easy to get it back:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ alias rm='rm -i'
skristja@asb9838n-a07:~/sfuhome/Demo$ alias rm
alias rm='rm -i'
```

The proper place to put the unalias command is in your `.bashrc` file, where it will take effect automatically. This is also a good place to add aliases you like or modify your default search path. The search path is the set of directories that Unix searches when looking for executable files.

What about deleting a directory? Let's move up out of the `Demo` directory and delete it:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ cd ..
skristja@asb9838n-a07:~/sfuhome$ rm Demo
rm: cannot remove `Demo': Is a directory
```

Hmmm . . . not very successful. Some of you may know that there is another command, `rmdir`, specifically for removing directories:

```
skristja@asb9838n-a07:~/sfuhome$ rmdir Demo
rmdir: failed to remove `Demo': Directory not empty
```

We could remove `Demo/himom.txt` and then remove the `Demo` directory, but that's not really necessary:

```
skristja@asb9838n-a07:~/sfuhome$ rm -r Demo
rm: descend into directory `Demo'? y
rm: remove regular file `Demo/himom.txt'? y
rm: remove directory `Demo'? y
skristja@asb9838n-a07:~/sfuhome$ ls
bin      CMPT 150     personal  $RECYCLE.BIN
cmpt125  desktop.ini  pub_html
```

The '`-r`' flag ('recursive') tells the `rm` command that you intend to remove the directory *and its contents, recursively*. The interactive alias makes this more than a little tedious. Use this command with care[4] if you have removed the interactive alias for `rm`!

The '`-r`' flag can be used in many commands, with the same meaning: Perform the action for the directory and all its contents, recursively.

In general, Unix command line applications take the attitude that you know what you are doing. They do not ask 'Are you sure?', they just do what you request. Think before you hit the return key.

How Do I Move Things Around?

Typically, you will want to copy or move files and directories. Recreate the `Demo` directory and `himom.txt`.

```
skristja@asb9838n-a07:~/sfuhome$ mkdir Demo
skristja@asb9838n-a07:~/sfuhome$ cd Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ echo 'Hi, Mom!' > himom.txt
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
```

[4] Some of you may be aware that the name of one of the university's mail servers is rm-rstar. Now you can appreciate the joke. Never type 'rm -r *' without careful thought to the consequences!

```
himom.txt
```

The cp ('copy') command will copy a file:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ cp himom.txt another.txt
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
another.txt  himom.txt
```

You can copy entire directory trees with a single command using the '-r' flag:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ cd ..
skristja@asb9838n-a07:~/sfuhome$ cp -r Demo Demo2
skristja@asb9838n-a07:~/sfuhome$ ls
bin      CMPT 150  Demo2        personal  $RECYCLE.BIN
cmpt125  Demo      desktop.ini  pub_html
skristja@asb9838n-a07:~/sfuhome$ ls Demo
another.txt  himom.txt
skristja@asb9838n-a07:~/sfuhome$ ls Demo2
another.txt  himom.txt
```

The mv command ('move') will move a file or directory from one place to another. To rename a file, simply move it to the new name:

```
skristja@asb9838n-a07:~/sfuhome$ mv Demo/himom.txt Demo2/byemom.txt
skristja@asb9838n-a07:~/sfuhome$ ls Demo
another.txt
skristja@asb9838n-a07:~/sfuhome$ ls Demo2
another.txt  byemom.txt  himom.txt
```

Moving a directory works the same way:

```
skristja@asb9838n-a07:~/sfuhome$ mv Demo2 NewDemo
skristja@asb9838n-a07:~/sfuhome$ ls
bin      CMPT 150  desktop.ini  personal  $RECYCLE.BIN
cmpt125  Demo      NewDemo      pub_html
skristja@asb9838n-a07:~/sfuhome$ ls NewDemo
another.txt  byemom.txt  himom.txt
```

As mentioned before, Unix assumes you know what you're doing. If you move or copy a file, and a file of the same name already exists, it will be overwritten without complaint:

```
skristja@asb9838n-a07:~/sfuhome$ ls Demo
another.txt
skristja@asb9838n-a07:~/sfuhome$ ls NewDemo
another.txt  byemom.txt  himom.txt
skristja@asb9838n-a07:~/sfuhome$ mv NewDemo/himom.txt Demo/another.txt
skristja@asb9838n-a07:~/sfuhome$ ls Demo NewDemo
Demo:
another.txt

NewDemo:
another.txt  byemom.txt
```

Be careful! If you would rather have a safety net, you can use the '-i' option for mv and cp[5]:

```
skristja@asb9838n-a07:~/sfuhome$ ls Demo
another.txt
skristja@asb9838n-a07:~/sfuhome$ ls NewDemo
```

[5] The current default aliases in CSIL provide 'cp -i' but not 'mv -i'.

```
another.txt  byemom.txt
skristja@asb9838n-a07:~/sfuhome$ mv -i Demo/another.txt NewDemo
mv: overwrite `NewDemo/another.txt'? n
skristja@asb9838n-a07:~/sfuhome$ ls Demo
another.txt
skristja@asb9838n-a07:~/sfuhome$ ls NewDemo
another.txt  byemom.txt
```

As you can see, typing 'n' (no) to the prompt aborts the move. If you like this style, create aliases.

Notice that the `mv` command specifies a specific file to move (`Demo/another.txt`) and only a directory (`NewDemo`) for the target. This says 'move the file to the target directory; but do not change the file name.'

Recall that I mentioned that file and directory names with spaces are awkward. If you forget, or you transfer a file from Windows and the name contains spaces, you must use quotes (single or double) around the name:

```
skristja@asb9838n-a07:~/sfuhome$ cd Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ touch "Name With Spaces"
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
another.txt  Name With Spaces
skristja@asb9838n-a07:~/sfuhome/Demo$ rm Name With Spaces
rm: cannot remove `Name': No such file or directory
rm: cannot remove `With': No such file or directory
rm: cannot remove `Spaces': No such file or directory
skristja@asb9838n-a07:~/sfuhome/Demo$ rm "Name With Spaces"
rm: remove regular empty file `Name With Spaces'? y
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
another.txt
```

Unix command shells assume that spaces separate parameters[6]. Here, the shell interprets '`Name With Spaces`' as three separate file names, and complains that it can't find any of them.

How Do I See What Is In a File?

In Unix, the `file` command will tell you what is in a file:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ cd ~/sfuhome
skristja@asb9838n-a07:~/sfuhome$ ls
bin      CMPT 150  desktop.ini  personal  $RECYCLE.BIN
cmpt125  Demo      NewDemo      pub_html
skristja@asb9838n-a07:~/sfuhome$ file Demo
Demo: setgid directory
skristja@asb9838n-a07:~/sfuhome$ file desktop.ini
desktop.ini: Little-endian UTF-16 Unicode text, with CRLF, CR line
terminators
skristja@asb9838n-a07:~/sfuhome$ file /bin/bash
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0xe643cefb2c672ad94e955067c511537ddbab48da, stripped
```

It is a good idea to use common extensions for files (e.g., `.pdf` for a PDF file) but the `file`

[6] So does the Windows command shell cmd.exe, but most people never use cmd.exe because it has so little power. If you want to learn a decent shell for Windows, have a look at PowerShell, included in Windows 7.

command does not rely on the file name extension. It looks at the content of the file[7] and makes its best guess, but it can be fooled.

Text files can be dumped to the terminal with the `cat` command, but usually you want a bit more control. The best pager is a command called `less` (a pun at the expense of the less capable pager program called `more`).

```
skristja@asb9838n-a07:~/sfuhome$ less Demo/another.txt
Hi, Mom!
Demo/himom.txt (END)
```

Type 'q' to exit the pager.

For serious file manipulation, you will want to use an editor. The two historical command line ascii text editors are `vim` and `emacs`. You will want to experiment with them. They have radically different approaches to editing, and the choice between them is entirely a matter of personal taste. The learning curve is steep for both of them, but they are very powerful when it comes to creating and editing program text[8]. There are many other editors available — ask friends, do some Internet searching, find something that works for you. Eclipse has a built-in editor with considerable knowledge of Java syntax and formatting.

To view PDF and Postscript, use the `evince` viewer. Adobe `acroread` is also available for PDF files. An easy way to read PDF in Ubuntu is to use the graphical file browser, and simply double click on the PDF file you want to open.

How Do I Compile and Run a Java Program?

Now that you know how to navigate the file system from the command line, the only remaining essential skill is how to compile and run a Java program.

- If you still have the HelloWorld project in the Cmpt125 workspace from the first demo, you can use it here. Starting in the sfuhome directory:

```
skristja@asb9838n-a07:~/sfuhome$ cd cmpt125
skristja@asb9838n-a07:~/sfuhome/cmpt125$ ls
HelloWorld
skristja@asb9838n-a07:~/sfuhome/cmpt125$ cd HelloWorld/
skristja@asb9838n-a07:~/sfuhome/cmpt125/HelloWorld$ ls
bin  src
skristja@asb9838n-a07:~/sfuhome/cmpt125/HelloWorld$ cd src
skristja@asb9838n-a07:~/sfuhome/cmpt125/HelloWorld/src$ ls
HelloWorld.java
```

  As you can see, each Eclipse project is a subdirectory in the workspace. The Java source file is in the project subdirectory. Copy it to the `Demo` directory, then `cd` to the `Demo` directory.

```
skristja@asb9838n-a07:~/sfuhome/cmpt125/HelloWorld/src$ cp HelloWorld.java ~/sfuhome/Demo
skristja@asb9838n-a07:~/sfuhome/cmpt125/HelloWorld/src$ cd ~/sfuhome/Demo
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
```

[7] More accurately, `file` looks at the initial bytes of a file.

[8] To take full advantage of the power of vi or emacs, and many other Unix command line programs, you will want to learn regular expressions, a powerful way of expressing text patterns. 'man 7 regex' will get you the man page for regular expressions. For a gentler introduction, search 'regular expression tutorial' in your favourite search engine.

```
another.txt  HelloWorld.java
```

- If you have deleted the HelloWorld project, you can download `HelloWorld.java` from the url `http://www.cs.sfu.ca/CourseCentral/125/skristja/Demos/CmdLine/HelloWorld.java`

  Place it in the `Demo` directory.

To compile a Java program, run the Java compiler, `javac`:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
another.txt  HelloWorld.java
skristja@asb9838n-a07:~/sfuhome/Demo$ javac HelloWorld.java
skristja@asb9838n-a07:~/sfuhome/Demo$ ls
another.txt  HelloWorld.class  HelloWorld.java
skristja@asb9838n-a07:~/sfuhome/Demo$ file HelloWorld.class
HelloWorld.class: compiled Java class data, version 50.0 (Java 1.6)
```

Running the compiler produces the file `HelloWorld.class`, which is Java bytecode. The compiler requires that you specify the `.java` extension on the source file.

To run the program, you run the Java interpreter java with the class file as the parameter:

```
skristja@asb9838n-a07:~/sfuhome/Demo$ java HelloWorld
Hello, World!
```

Do not specify the .class extension! The Java interpreter will complain. (It tries to interpret the '.' in terms of Java's notion of how derived class files should be stored in the file system.)

During the semester, you will be asked to submit your Java source files as your solution to an assignment. As explained above, you can find the source files within the project directory in the Eclipse workspace.

Scripts

One of the great strengths of using the Linux command line interface is that it provides you with the ability to automate tasks. To do so, your can create bash scripts which will perform customized functions for you. We will discuss that in an upcoming tutorial.

Check in with the TA

Show the TA your HelloWorld.java compiled and running from the command line before you go.

I Want More!

If this is already more command line Unix than you wanted to know, you can stop now. This tutorial contains enough to allow you to do the work required for the course. If you're looking for more information, there's lots of it out there. Consult the Unix documentation links at the bottom of the Resources page on the course web site.

B. Lab Exercises – To be completed by Students

Student are responsible for having the TA look at your work by end of tutorial for full marks. Marks will be awarded for attendance and for making progress on the Lab Exercises. The exercises do not need to be completed by the end of the lab for full attendace/participation marks, but some progress needs to be shown.

Students may work in teams of two and complete the exercises as a team.

1. **Ensure you are running Eclipse in a Java Perspective**
   a. Check if the upper right corner shows a box with JAVA written in it.
   b. If it does not say Java, you are in the wrong perspective. Follow the next step to set your perspective to Java
   c. Window → Open Perspective → Other… → Java
2. **Create a new java project called Lab03Strings**
3. **File → New → [Select New Project] → [Project Name: Lab03Strings] → Finish**
   a. You will see a new Project called Lab03Strings in the Package Explorer. Ensure that it is highlighted by selecting, if necessary, by clicking on it.
4. Create a new java class called Lab03Strings
   a. File → New → [Select New Class] → [Class Name: Lab03Strings]
   b. Be sure to select the box labeled “Public static void main” in order to create your public method called main. This is where your program starts .
   c. Click on *Finish* to create the Class file named `Lab03Strings.java`
   d. This will create a new file in the edit window called Lab03Strings
5. Edit your `main` method to contain the following code inside the method body, which will declare and initialize the strings s1, s2, and s3 then do some `println’s`. Observe the results.

```
String s1 = "this is my first string 1,2,3,4 – bye";
String s2 = "** second! **";
String s3 = "And a third string!!!";
System.out.println(s1 + s2);
System.out.println(s1.toUpperCase() + "** ohhh! ");
System.out.println("Third String is \"" + s3 + "\"");
System.out.println("the length of s1 is: " + s1.length());
```

   a. Run the code and observe:
      i. s1.toUppercase() indicates that you are calling the method toUppercase through the object string s1 (and you are not passing any parameters, as there is an empty list of parameters, as indicated by (). What does the method return?
      ii. s1.length() indicates that you are calling the method length through the

object string s1 (and you are not passing any parameters, as there is an empty list of parameters, as indicated by (). What does the method return?

6. Explore what happens when you assign one String to Another. Add the following code at the end of your existing main program:

```
System.out.println("s1 is \"" + s1 + "\"");
System.out.println("s3 is \"" + s3 + "\"");

System.out.println("Assigning s3 = s1;");
s3 = s1;
System.out.println("s1 is \"" + s1 + "\"");
System.out.println("s3 is \"" + s3 + "\"");

System.out.println("Concatenating onto s1...");
s1 += " has changed";
System.out.println("s1 is \"" + s1 + "\"");
System.out.println("s3 is \"" + s3 + "\"");
```

Run the new code and observe which has changed. What is going on? Notice that even though you set s3 to point to the same string object as s1, changes to s1 are not reflected in s3. Why? Because Strings are immutable. Modifying s1 creates a new String object instance and assigns a reference to it in s1. String object reference s3 still points to the original String object.

7. Compiling with the Linux Command Line

Remote access to graphical user interfaces such as eclipse is not possible in CSIL. It has been disabled. You can still login and test your code from home, but you need to be able to do it via the Linux command line. Let's practice creating and running a program from the command line.

a. Start up a command line terminal session and go to your `Cmpt125` directory

```
skristja@asb9838n-a07:~$ cd ~/sfuhome/Cmpt125
```

b. Create a new directory for lab03str and go there

```
skristja@asb9838n-a07:~/sfuhome/cmpt125$ mkdir lab03str
skristja@asb9838n-a07:~/sfuhome/cmpt125$ cd lab03str
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03str$
```

c. Copy your `Lab03Strings.java` file from your eclipse project folder to here: (I will leave out the Linux prompt so the `cp` command fits on one line)

```
cp ~/sfuhome/cmpt125/Lab03String/src/Lab03Strings.java .
```

d. Now just like in the Lab Demo, compile and run your java program from the command line and you should see the same output as before:

```
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03str$ javac Lab03Strings.java
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03str$ java Lab03Strings
this is my first string 1,2,3,4 b bye** second! **
THIS IS MY FIRST STRING 1,2,3,4 B BYE** ohhh!
Third String is "And a third string!!!"
the length of s1 is: 38
```

```
s1 is "this is my first string 1,2,3,4 b bye"
s3 is "And a third string!!!"
Assigning s3 = s1;
s1 is "this is my first string 1,2,3,4 b bye"
s3 is "this is my first string 1,2,3,4 b bye"
Concatenating onto s1...
s1 is "this is my first string 1,2,3,4 b bye has changed"
s3 is "this is my first string 1,2,3,4 b bye"
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03str$
```

e. You can edit files remotely using any command line editor such as vi, vim, or emacs. Let's try vim:

```
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03str$ vim Lab03Strings.java
```

f. You will enter an edit session with `Lab03Strings.java` in the main window. To exit without changing anything, type the following three characters exactly:

```
:q!
```

g. At this point, you have not made any changes to `Lab03Strings.java`, but you have experienced the vim command line editor. When you enter VIM, it expects that every character you type is a command that takes effect immediately. One such command is the "`:`" character that moves your focus immediately to the VIM command line at the bottom where it expects a command. The command you typed was "`q!`" which means quit and discard all unsaved changed. This is a good command to know!

h. Some other useful commands within vim allow you to append, insert, or replace characters as you type. You will stay in this mode until you type the `<escape>` key on the upper left of the keyboard. The most used such commands include:

```
A  – Append any typed characters to the end of a line
i  - Enter Insert mode and insert characters as you type
R  - Enter Replace mode and Replace characters with the ones being typed
```

Give it a try but remember to hit `<escape>` to get out of these modes. Then type `:q!` to abort any changes that you have made.

i. Some commands take immediate affect but leave you in command mode so you there is no need to hit <escape> to return to command mode. These commands include:

```
x  - delete current character
dw – delete to end of current word
dd – Delete current line
r  - replace current character
.  – repeat the last command
```

For these commands, you can optionally enter a number before the command, then the command will be repeated that many times. For example, this command will delete the next 10 lines:

```
10dd
```

j. You may want to cut and paste code. You can cut the code using the vim `dd` command. Then move the cursor to where you want to paste the code, and type the letter `p` to paste the code after the current line.

k. Command line editing is not for the faint of heart, but learning it is worth the trouble. Until you have mastered `vim`, you can always use a graphical editor when you are sitting at a CSIL computer or at your home PC, but your choices are limited to command line text based editors like `vim` when you login to CSIL remotely. Fortunately, VIM provides an extensive help menu. To view the `vim` help menu, use the `help` command:

```
:help
```

l. Later, you can save your changes using the write and quit command: `:wq` For now, discard all your changes by typing `:q!`

```
:q!
```

m. Make a copy of your original `Lab03Strings.java` program as the new java file `Lab03Strings2.java`. Edit the new copy of your file using vim.

```
cp Lab03Strings.java Lab03Strings2.java
vim Lab03Strings2.java
```

n. Within the editor, the first change you need to make in order for this to compile is to change the Class name to match the new file name. Within VIM, move the cursor with the arrow keys until the cursor is immediately after the lowercase `s` on the first line. Type the following three keystrokes to append the character `2` to the class name. Remember that <escape> represents hitting the <escape> key on the keyboard.

```
i2<escape>
```

The first line should now look like:

```
public class Lab03Strings2 {
```

Type the following three keystrokes to save your changes and exit vim:

```
:wq
```

o. Now compile the newly edited version of `Lab03Strings2.java`. The output should match the original.

```
javac Lab03Strings2.java
java Lab03Strings2
```

p. If you want to use CSIL remotely from home, you will need to get familiar with `vim` so you can edit your files remotely. You will use `ssh` to login remotely and `scp` to transfer files to a CSIL machine. If you prefer, you can always edit your files at home using a graphical editor and then use `scp` to copy the new version back to your CSIL machine so you can test it. The Course Webpage has links describing how to access CSIL remotely. We will look into that in more detail in another lab.

8. Explore: the 'for-loop' in java

Java allows code to is executed a number of times by including it as the body of a for-loop. The idea of a `for` loop in Java is similar to a `for` loop in Python and other languages, but there are some syntactic and subtle differences.

a. Read section 4.8 of the text

b. Create a test java file called Lab03For.java using either Eclipse or the command line based on the Stars program from Listing 4.12 of the text. If using Eclipse, start by creating a new Java Project called Lab03For, then the Class Lab03For. If using the Linux command line, you just need to copy this text into a Lab03For.java file.

```
//********************************************************************
//  Lab03For.java
//  From Stars.java by Java Foundations
//
//  Demonstrates the use of nested for loops.
//********************************************************************
public class Lab03For // Class Name changed from Text to Match filename
{
   //-----------------------------------------------------------------
   //  Prints a triangle shape using asterisk (star) characters.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int MAX_ROWS = 10;

      for (int row = 1; row <= MAX_ROWS; row++)
      {
         for (int star = 1; star <= row; star++)
            System.out.print ("*");

         System.out.println();
      }
   }
}
```

c. Compile and run the Lab03For program to watch a Java `for` loop in action.

```
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03for$ javac Lab03For.java
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03for$ java Lab03For
*
**
***
****
*****
******
*******
********
*********
**********
```

d. Nesting `for` loops

What does the following code do? First, try to figure this out on paper and pencil

first. Then test it inside the main method in the `Lab03For` application from the previous question. Insert this code in the `main` method body after the } following the body of the existing `for` loop.

```
for (int i = 1; i<= 2; i++)
{
   for (int j = 1; j<=4; j=j+1)
   {
      System.out.println("i is:" + i + " and j is " + j);
   }
   System.out.println( );
}
```

9. Explore the Scanner Class

a. The `Scanner` methods allow us to interact with the user. The `Scanner` class has to be imported. A `Scanner` object needs to be created since we need to *instantiate* a `Scanner` object instance to read from a specific input stream such as System.out.

b. Create a new project called `Lab03Scanner` and create `Lab03Scanner` Class with a `public static void main()` method as before.

c. Within the Java file, you will need to insert the following line to import the java.util.Scanner package. Insert it after the block comment and before the class definition.

```
import java.util.Scanner;
```

d. We are going to re-create the `GasMileage` example from Listing 2.9 of the text. To compute mileage, in miles/gallon, one needs to take the number miles travelled and divide by the number of gallons of gas used to get there.

e. To start, we will need an integer variable to represent the number of miles that we have driven. Define an `int` at the start of your `main` method body called `miles`.

f. Next we need to know how many gallons of gas our car used. This is not likely to be a whole number so it should be represented as a floating point number. Define a second variable called gallons which is a double.

g. Finally, the computed mileage will be a fractional value so also needs to be stored in a double. Define it as the third local variable in your method.

h. In this program, we want to use the `Scanner` class to allow us to receive input from the user. Declare a Scanner object called `scan` and instantiate it by called the `Scanner` construction method and specifying that the input stream is the `System.in` object:

```
Scanner scan = new Scanner (System.in);
```

i. Now you use the scan object to request an `int` value from the user about how far they travelled. Use the `nextInt` method to read the input stream and return an integer value:

```
System.out.print ("Enter the number of miles: ");
miles = scan.nextInt();
```

j. Try compiling and running your program. Does it behaive as expected? Were you able to input the number of miles travelled? If running from the command line, run the program and wait to be prompted for input. Simply type an integer value like `100` and then hit the <enter> key. If you are running this from Eclipse, you will need to click on the console window before you start typing your input value and then hit <enter>.

k. You are not finished, your program needs to read in the amount of gas used and compute the mileage. See if you can figure out how to do this on your own. You will need to use Scanner's `nextDouble` method at some point to read in the number of gallons of gas used. Then compute the mileage in variable mpg as follows:

```
mpg = miles / gallons;
```

l. Finally, print out the computed mileage using:

```
System.out.println ("Miles Per Gallon: " + mpg);
```

m. Compile and test your completed program. Does it work as expected? Test it with various input values. What happens if you enter a decimal number like 2.5 as the number of miles travelled? Can you make your code more robust so that it checks for this and then re-requests that the user input integer data only? What if the user enters non-numeric data like "Crash"?

10. READ: Brief introduction to static and non-static methods

We talked about this briefly in class regarding the difference between static and non-static methods. Here is a little more detail. This will help you understand the following exercises.

In Java, there are methods that are associated with an object when executed. These are called ***non-static*** methods. For example, `charAt(n)` is a `String` non-static method whose execution is associated with a specific `String` object instance using the "dot" (.) notation. As an example, `firstLetter` will be the first character (position 0) that the `String` wordVariable contains. The method `charAt` is invoked through `wordVariable`

```
String wordVariable = "abcde";
char firstLetter = wordVariable.charAt(0); // non-static method invocation
```

There are also ***static*** methods. They can be thought of as "useful functions", collected together in some class or module. Static methods receive all the needed information in the parameters. They are not associated with any object but instead are associated with the class. For example, `sqrt(n)` is a static method in the `Math` class to calculate the square root of the parameter `n`. One should include the class name using the "dot" notation.

```
double result = Math.sqrt(25);
```

11. Explore the Math Class

a. Do some calculations using various `math` methods. Since its methods are static, you do not create an object of the class Math to use the method but instead you qualify the method with the class name using the dot notation. See section 3.5 in the textbook.

b. Create a new project called Lab03Math with a Lab03Math Class

c. In the main method, you will re-create the Quadratic example from Listing 3.3 in the textbook to compute the roots of a Quadratic equation using methods from the Math class. Start by including the following statements to declare your three integer quadratic coefficients a, b and c at the start of the `main` method body.

```
int a, b, c;  // ax^2 + bx + c
```

d. Next, you need to have two doubles for storing the computed roots:

```
double root1, root2;
```

e. As with the Scanner exercise, instantiate a Scanner object and request the input values from the user:

```
Scanner scan = new Scanner (System.in);

System.out.print ("Enter the coefficient of x squared: ");
a = scan.nextInt();
System.out.print ("Enter the coefficient of x: ");
b = scan.nextInt();
System.out.print ("Enter the constant: ");
c = scan.nextInt();
```

f. To compute the discriminant, you need to compute b^2 and subtract 4ac. The static `Math` method `pow` can compute b^2 or you could simply multiply b by itself. Notice that the method is being called without an object reference. That works because it is a static method.

```
discriminant = Math.pow(b, 2) - (4 * a * c);
```

or

```
discriminant = (b * b) - (4 * a * c);
```

g. Do you recall how to compute the roots? You will need to compute the square root of the discriminant. Use the `Math.sqrt` static method for that:

```
root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
System.out.println ("Root #1: " + root1);
System.out.println ("Root #2: " + root2);
```

h. Try to compile and test it. Does the compile fail? What is missing? Recall that the Scanner Class is not part of java.lang so it needs to be imported. Add an `import` statement before the Class definition as follows:

```
import java.util.Scanner;
```

i. Does the compiler complain about discriminant not being defined? Remember that you are not using Python anymore, in Java you need to declare all variables before you use them. Try to figure it out on your own and make it work: it is good practice. Fix the code and test it to make sure it

runs over all reasonable values.

j. Here is an example output with a=1, b=0, c=-1:

```
skristja@asb9838n-a07:~/sfuhome/cmpt125/lab03$ java Lab03Math
Enter the coefficient of x squared: 1
Enter the coefficient of x: 0
Enter the constant: -1
Root #1: 1.0
Root #2: -1.0
```

12. Explore the Random Class

a. Create a new project called Lab03Random and create a new Class called Lab03Random within it.

b. The `Random` Class is an example of a non-static interface. That means that calls to its methods require a reference to an object instance. You will need to construct a `Random` object instance since the methods are non-static. Notice in particular the (non-static) method `nextInt()` taken from listing 3.2 of the text which generates a random integer from `MIN_INT` to `MAX_INT`:

```
int num1;

num1 = generator.nextInt();
System.out.println ("A random integer: " + num1);
```

c. In the above code fragment, `generator` is an instance of the `Random` Class. It is an object which is responsible for initializing the random seed when it is constucted, and for generating random numbers when one of its methods, such as `nextInt`, is invoked. Before you can use the generator, you will need to instantiate it:

```
Random generator = new Random();
```

d. Notice that before this will compile, you need to import the Random class to be able to use its constructors and methods. You do so with the statement:

```
import java.util.Random;
```

e. Include the import statement before the class header

f. You can generate random numbers in a range from 0 to N by invoking method nextInt with a range parameter. If you want a number from 0 to 9, use the following:

```
num1 = generator.nextInt(10);
```

g. To generate a different range, you will need to do the math yourself. The following generates random numbers in different ranges:

```
num1 = generator.nextInt(10);      // From   0 to 9
num1 = generator.nextInt(10) + 1;  // From   1 to 10
num1 = generator.nextInt(15) + 20; // From  20 to 34
num1 = generator.nextInt(20) - 10; // From -10 to 9
num2 = generator.nextFloat();      // Random float between 0 and 1
```

```
      num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
```

h. Experiment with generating different random ranges and printing the results. Retain each random range and println that you generate in your final version so the markers can see your work. Compile and test your application. Run the Lab03Random program several times to see how the numbers change with each run.

i. Your final version of Lab03Random should be well commented and contain as a minimum the following random ranges:

```
//********************************************************************
//  RandomNumbers.java       Java Foundations
//
//  Demonstrates the creation of pseudo-random numbers using the
//  Random class.
//********************************************************************
import java.util.Random;
public class RandomNumbers
{
   //-----------------------------------------------------------------
   //  Generates random numbers in various ranges.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Random generator = new Random();
      int num1;
      float num2;

      num1 = generator.nextInt();
      System.out.println ("A random integer: " + num1);

      num1 = generator.nextInt(10);
      System.out.println ("From 0 to 9: " + num1);

      num1 = generator.nextInt(10) + 1;
      System.out.println ("From 1 to 10: " + num1);

      num1 = generator.nextInt(15) + 20;
      System.out.println ("From 20 to 34: " + num1);

      num1 = generator.nextInt(20) - 10;
      System.out.println ("From -10 to 9: " + num1);

      num2 = generator.nextFloat();
      System.out.println ("A random float (between 0-1): " + num2);

      num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
      num1 = (int)num2 + 1;
      System.out.println ("From 1 to 6: " + num1);
   }
}
```

C. Lab Exercise Submission – To be completed by Students

Student are responsible for submitting the requested work files by the stated deadline for full marks. Since Lab Exercise solutions will be discussed in class following the submission deadline, late submissions will NOT be accepted. It is the student's responsibility to submit on time.

Students may work in teams of two and submit a single set of files on behalf of the group in Canvas.

1. **You must submit your final version of the following files before the deadline. Students must ensure that all submitted code compiles and is properly commented and formatted for readability:**
   - `Lab03Strings.java`
   - `Lab03Scanner.java`
   - `Lab03Math.java`
   - `Lab03Random.java`
2. **For students working in the lab with a partner, only one submission is required for the group of two students**
3. **Files are to be submitted into CourSys under Lab03. Use the Manage Groups menu to create a group name unique to you and which includes the week number. See the course website for submission instructions at http://blogs.sfu.ca/courses/fall2013/cmpt125/labs/submitting/**