APS 101 — Computer Programming

Basic UNIX, Assignment submission, working from home, and more.

Winter 2008

Introduction

This manual serves as your introduction to the computers we will be using in this course. Rather than running in a Windows or Macintosh environment that you may know, our computers are running a variation of UNIX called Linux, specifically the version produced by Red Hat. As you will see, UNIX is significantly different from the Windows or Macintosh environments.

This document is in the form of a tutorial that will gradually take you through the basics of interacting with the computer. You are also encouraged to continue your introduction to the computing environment by reading the manual *Getting Started With UNIX*. It can be found online at

`http://www.ecf.utoronto.ca/ecf/docs/unix`

The X Window Environment

After logging in, you will be presented with a sequence of windows. After reading each one, click on the `okay` button in the lower left corner of the window to make it go away (or just wait and it will eventually disappear on its own). Red Hat Linux will then start up. When the system starts up, there will be a *taskbar* along the bottom of the screen. There may also be windows or icons on the screen. When you log out, the desktop is saved so that the environment is saved for your next login session.

Most of your work will be done using *terminal windows*. Start a new terminal window by clicking the red hat button in the taskbar. A menu will pop up. Choosing System Tools will cause another menu to pop up. In the second menu, click on Terminal to open a terminal window. The window can be moved around on the screen with the mouse by pressing on the title bar (along the top of the window) and dragging it to another location. Give this a try.

You can also change the size of a window using the mouse. Move the cursor to the edge of the window. You should see the cursor change to a double-headed arrow. To make the window bigger or smaller, press the mouse button, drag the cursor in the desired direction, and then release the mouse button.

At the top right of the window are three little boxes to control the window. The × closes the window, the middle one maximizes the window size, and the left one makes it disappear. To make the window reappear, click on its name in the taskbar.

In the terminal window, you should see the UNIX prompt which may vary but will usually end with the character `%`. We will show the prompt simply as `%`. You can issue commands directly to the UNIX operating system at this prompt. The first command you will enter will tell UNIX to create a new window. Move the pointer with the mouse so that it is on top of your terminal window and enter the following command:

```
% xterm &
```

The `xterm` command stands for X Terminal and it creates a window that, like the original terminal window, has a UNIX prompt at which you can enter commands. You can now use either the original terminal window or the new xterm window to issue UNIX commands. You can create additional windows with the `xterm &` command if you like and have several windows on the screen.

Although there can be many windows on the screen, only one window is active at a time. The active window has a highlighted title bar (along the top edge), and all text typed at the keyboard is directed to the active window. You can make another window active by clicking the title bar, the same way Microsoft Windows works. More advanced UNIX users change their settings so having the mouse pointer over a window is sufficient to make it active. To change your setup to this method, you will have to change the focus behaviour of your desktop and window manager.

When you create new windows with the `xterm` command, the ampersand (`&`) is an optional component of the command. By default, commands are executed in the foreground. Adding an ampersand to the end of a command instructs UNIX to execute it in the background. Commands that are executed in the foreground must complete execution before the command line that started it can do anything else. So if you call up a new window using the `xterm` command and do not include an ampersand, you will not be able to use the window from which you called up the new one until the new xterm window is closed. When a command is executed in the background, the command line from which it was called can continue to accept commands while the background command is executing. When calling up new windows from the command line, you should always add an ampersand at the end of the command so that you can use both the new and the original windows.

Changing Your Password

At this point an important task that you should perform is to change your password. Remember that the password to your account is the only security measure keeping your account private so you should choose your password carefully and keep it an absolute secret. You change the password by using the `passwd` command. Your password should be at least 6 characters long, with 7 or 8 being preferable. Try to use a mixture of numbers and letters in both upper and lower case, making it less likely that someone will be able to guess your password. In particular, do not use any names or single words that could be found in a dictionary. Since computers are good at tedious tasks, people have written programs that can try to guess your password using a dictionary and other guessing schemes. A good way to create a password is to take a sentence that is easy to remember and use the first letter of each word or a numeral for words like "to"(2) and "for"(4). For example, the sentence "I hate to be late for computer class" could be used to produce the password Ih2bl4cc.

When changing your password, a new window on skule.ecf, the main server, may start up. The new password dialog should look something like this:

```
% passwd
opening window on skule for password change
Changing password for <your login name>
Old password: <type in your current (old) password here>
New password: <type in your new password here>
Re-enter new password: <type in your new password again>
```

As you type both the old and new passwords, the characters will not be visible on the screen. This is deliberate, to prevent others from seeing your password by looking over your shoulder. Because you cannot see your password, you are asked to type it twice, on the theory that you are unlikely to make the same typing mistake twice.

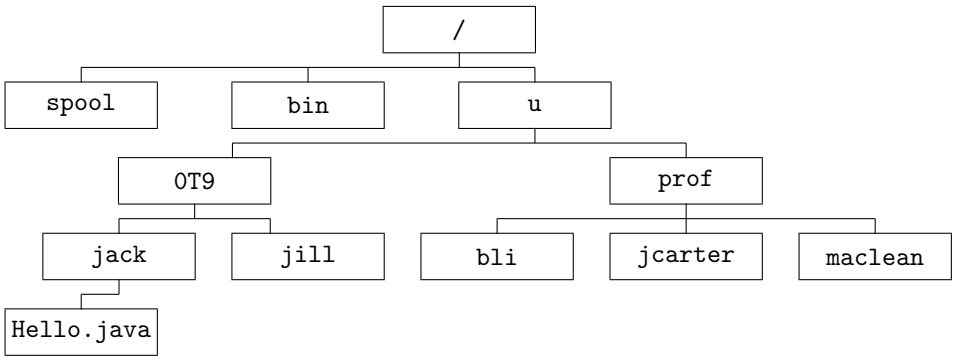
File Organization

Data and programs that you use are stored as files on a disk. There are many types of files: some are software programs that can be executed, others contain text that can be read, while others contain images that can be displayed on the screen. Two things that all files have in common are that they can be stored in computer storage and that they all have a name. We will soon take a look at some files on the ECF computer system.

As we add more files to computer storage the number of files grows to the point where it is hard to keep track of them all. For this purpose, files are organized into *directories*. You can think of directories as being like file folders. File folders are used to organize papers, articles, and other things and are named according to their contents. You may well be familiar with the concept of folders from Windows or Macintosh systems. In UNIX, folders are called directories.

If you mapped out the directory and file organization onto a graphical representation, you would get a tree-like structure as shown in the diagram. At the top is the *root directory*. It is represented by a slash character, which is its name. The root directory may contain files, and it will also contain other directories. (We say that the root directory is the *parent* of these files and directories.) The

files and directories within the root directory are shown as nodes below and attached to the root directory. These directories may have files and other directories contained within them and these are shown as nodes below and attached to their parent directory. This pattern can be repeated many times.



UNIX File Organization

It is the files that are actually important when it comes to running programs, whereas the directories are only there to organize the files. When you want to work with files, you generally have immediate access to the files in one directory only. This directory is called the *working directory*. To find out what the present working directory is, use the `pwd` (print working directory) command by typing the following:

```
% pwd
```

UNIX will respond with a string that looks like `/u/0T9/yourLogin`. This string describes a path through the directory tree from the root directory to the present working directory. Reading it from left to right, the path starts at the root directory, which contains the directory named `u`, which contains the directory `0T9`, which contains the directory with the same name as your login. What this string is saying is that there is a directory with the same name as your login name and it is located in the path described above and it is the present working directory.

You can change the present working directory using the `cd` (change directory) command. The `cd` command takes an *argument*, which is an extra term that is input after the command. In this case the argument is the path of the directory to which you want to move. There has to be a space between the command and the argument. Try the following:

```
% cd /
```

Then enter the `pwd` command. You will see that the present working directory is as you have requested — the root directory. Now try the following:

```
% cd /u
```

Entering the `pwd` command will show that you are now in the directory requested. Finally, enter:

```
% cd /u/0T9/yourLogin
```

and this will take you back to your home directory as you can verify using the `pwd` command once again.

Up until now, we have been specifying the path from the root directory to the desired directory as the argument to the `cd` command. This is called the *absolute path*. It is not necessary to always

give the absolute path as the argument and usually one does not do so. Rather, one can use what is known as a *relative path*.

To see how relative paths work, start by going to the root directory by entering `cd /`. Now go to the `u` directory by entering the following command:

```
% cd u
```

If you give a `pwd` command, you will see that you are now in `/u` even though you did not have the slash beginning the argument of the `cd` command. Because the argument did not begin with a slash, UNIX understood you to mean go to the `u` directory from the present working directory, which is `/`. Now try to go to the `0T9` directory by entering:

```
% cd 0T9
```

Again, if you enter the `pwd` command you will see that you have moved into the new directory.

There are two important short forms that are used to specify directories. Two consecutive periods (`..`) refers to the parent directory of the current directory, so if you use the command `cd ..` the parent directory will become the working directory. The other important short form is the ~ character, which refers to your home directory. No matter where you are in the directory tree, entering `cd ~` will make your home directory the working directory. With the `cd` command, you can get even simpler than that and enter just `cd` to make your home directory the working directory.

Practice using these short forms. First, enter a command to go directly to your home directory (`cd ~` or `cd`), then back up the directory tree, one directory at a time, using the `cd ..` command. Then return to your home directory once again.

Now you should have a good understanding of navigating around directories. Of course the interesting part is not directories, but what they contain. To determine what a directory contains, one uses the `ls` command, which stands for "list sorted" and instructs UNIX to list the contents of the directory, which will include any files and subdirectories. Now let us see what we can find using the `ls` command. Go to the root directory and enter:

```
% ls
```

You will see some strings arranged in columns. The `ls` command is displaying the names of files and subdirectories contained in the root directory. Try using `ls` in the `/u` directory and in other directories if you wish.

Now go to your home directory and enter `ls`. There is probably no output. This is because, as a new user, you have not created any new files or directories. However, even though `ls` does not show you any files in your home directory, there are some there that are hidden. To show *all* the files, including those that are hidden, enter:

```
% ls -a
```

Some filenames will be displayed. The names of hidden files are easily distinguishable as they all begin with a period. When your account was created, your home directory and a few hidden files were automatically created for you. These files are hidden because you would not normally work with them as you would with your program files. Hidden files usually contain special information for the computer system on various things such as how to set up your screen display or how to respond to your commands. You may, at some other time, want to explore some of these files and determine what they do.

One last variation on the `ls` command that you will probably find helpful is:

```
% ls -al
```

The final `l` (letter ell) stands for long form; entering this command will show all files and directories with more descriptive information for each item. A concise summary of directory and file organization can be found in the *Getting Started With UNIX* manual.

Manipulating Directories and Files

In the previous section you were shown how to explore the directory tree and how to look at existing directory contents. Now you will learn the basic commands for changing directory and file organization. For the most part, this will be constrained to directories and files within your own home directory.

Changing the directory tree structure is accomplished through the use of two basic commands: one that creates new directories and one that destroys them. The `mkdir` command makes a new directory. To illustrate its use, go to your home directory and create a new directory there by entering

```
% cd ~
% mkdir aps101
```

The argument supplied with this command is the name of the new directory: `aps101`. You can verify that a new directory was made by using the `ls` command. Notice that, by default, the new directory is placed within the present working directory.

Removing directories that are no longer required is just as easy. To do so, you go to the directory in which the unwanted directory is contained, then use the command:

```
% rmdir <directoryName>
```

Note that a directory cannot be removed unless it contains no files and no other directories.

Basic file manipulation is achieved through the commands that copy, move, and delete files. The `cp` (copy) command is used to make a duplicate of an existing file. It takes two arguments: the first argument is the name of the file to be copied; the second argument is the new name of the new, duplicate, file. In your home directory, enter the following command:

```
% cp .login new.file
```

This command has made a copy of the hidden file `.login` and called it `new.file` which is not a hidden file because its name does not begin with a period. Verify that the copy was made using the `ls` command.

The `mv` (move) command changes the name of a file and can also move a file into a different directory (an idea that will be discussed shortly). Try the following:

```
% mv new.file newname.file
```

Again, the `ls` command will show you the change that was made.

Finally, files can be deleted by using the `rm` (remove) command. Delete the new file that you created with the following:

```
% rm newname.file
```

Note:Be careful with `rm` command! If you remove a file it is deleted permanently and cannot be recovered.
[1ex]

Up until now, the file and directory manipulation commands have been demonstrated to operate only within the present working directory. These commands are in fact more versatile than this because it is possible to work with files and directories other than those in the present working directory. To work with a file or directory outside of the present working directory, you must supply its path rather than just its name for the command argument. Absolute or relative paths can be used.

For example, `cp` can be used to make a duplicate of a file that is not within the present working directory. Use the following command to copy the Java program `Hello.java` which is located in the directory `/share/copy/aps101s` to your home directory:

```
% cp /share/copy/aps101s/Hello.java .
```

The first argument of the `cp` command is the source file; here it contains the absolute path of the `Hello.java` program file. The second argument is a single period, a commonly used short form that represents the present working directory. UNIX interprets this command to mean: place a copy of the file `Hello.java` (located in `/share/copy/aps101s`) into the current directory and give it the same name, `Hello.java`. Now try using the `mv` command to place this file in the `aps101` directory you created earlier:

```
% mv Hello.java aps101/Hello.java
```

The `mkdir`, `rmdir`, and `rm` commands also can take paths as arguments. Experiment with these commands some more until you get the hang of it. Use the UNIX manual as a reference.

Examining the Contents of a File

We saw earlier that the `ls` command could be used to list the names of files. If we want to see the *contents* of a file, we can use the `cat` (which stands for catenate) command. To see the contents of the `Hello.java` file, change directory, if necessary, to the `aps101` directory and then give the command:

```
% cat Hello.java
```

For long files, the `cat` command is often not convenient because the first part of the file flies past your eyes too quickly. In such cases, you can use the `more` command. Execution of the command will cause one screenful of the file to be shown. To obtain the subsequent screenfuls, hit the space bar. To exit, hit the `q` key.

Creating, Compiling, and Running a Java Program

There are two approaches to creating, compiling, and running a Java program for this course. The first is to use a separate tool to perform each action: a *text editor* to create the program file, a compile command to compile it, and another command to run it. The second is to use an *Integrated Development Environment* (IDE) which allows you to perform many tasks from a single, usually graphical, program interface. The next three sections address both of these approaches.

Text Editing

The most basic and most important files that you will be working with are *text files*. A text file consists of readable text that can be created using the keys on a standard keyboard. Text files are used for various purposes so you will want to have a good knowledge of how to work with them. In this course, you will be making text files that will serve as Java programs.

You can create or make changes to a text file using what is called a *text editor*. There are several text editing programs available for use on the computer network. One of the simplest is called *nedit*,

and you should try using it now. To start the editor, use the command:

```
% nedit &
```

The nedit window that pops up differs slightly from the xterm windows that we have been using. This window is specifically designed for editing text files. There are two areas in this window that we need to understand. The first is a menubar along the top with categories of actions that may be performed. The second is a large empty box in the lower part of the window where the text file can be composed. The editor starts up with this section cleared so that you can create a file from scratch. Move the mouse so that the cursor is in this section. Then try typing something like `Testing 1, 2, 3, ...`

Once you have typed in your text, you can save it in a file. To do so, first click on the word File in the menubar, then click on Save As. In the window that pops up, type `test.txt` in the white space near the bottom. Finally, click OK. You have just created a new text file. Select File → Exit from the menubar to close the nedit window.

To see that a new text file was created, go to the xterm window and enter an `ls` command. The file `test.txt` should appear in the directory listing.

Open up an nedit window again using the command written above. As well as creating new files, nedit can be used to change existing files. To open an existing file, select File → Open from the menubar. Another pop-up box displays our directory and file listing. Select the file we have just created by double-clicking on `test.txt`. The file you created is loaded into the text file section. You can now edit the file. Point with the mouse just after the word `Testing` and click once. Now press the backspace key on the keyboard several times until the word is erased. Now, type in the word `Editing`. Notice that (modified) has been added to the window label next to the name of your file. This tells you that you have made changes to the file, but have not saved them. To save this file (which has already been named), select File → Save from the menubar. Finally, select File → Exit from the menubar to exit the editor.

If you explore the nedit menubar, you will discover features that will probably help you with future programming exercises. Search permits you to look for words, or replace them in your text. Preferences permits you to highlight the structure (syntax) of your Java programs and show your line numbers (useful for finding problems). Shell permits you to execute commands to compile your Java programs without exiting the editor — your compilation errors will be displayed in nedit so that you can make immediate corrections. The Shell options may be too difficult for you at first but, as your UNIX comfort-level increases, you may want to experiment with this.

Other editors are available in UNIX, directly from an xterm window. You may want to try out `pico` which, like `nedit`, is for beginners, and `vi` or `emacs`, which are for more advanced users.

Compiling and Running a Java Program

Earlier, you copied a Java program, called `Hello.java` into the `aps101` directory. The program, when it runs, prints the message `Hello, world!` in a window. To make it do this we have to go through two steps: first *compile* the program, a process that translates the program from Java into something called byte code, and then *run* the translated program. To compile the program, from a terminal window, go to the `aps101` directory and start the Java compiler by typing:

```
% javac Hello.java
```

You run this program by starting the Java interpreter as follows:

```
% java Hello
```

You should see the output of the program, saying `Hello world!`

An Integrated Development Environment — drjava

Program development is easier with a simple IDE like *drjava*. There are a number of ways of

launching drjava. Two of them are:

1. From a terminal window, type: `% drjava &`

   or

2. Click on the red hat in the lower left-hand corner of the screen, select ECF Menu and then select drjava.

In either case, the system will then display the drjava application window.

Before you can actually perform anything useful with this IDE, you may have to identify the location of certain files that it requires in order to compile and run your programs. To do this, select Edit → Preferences from the drop-down menu. The system displays the Preferences window. You must enter the location of the following two required files.

1. For the `Tools.jar Location`, enter: `/usr/java/j2sdk1.4.2_09/lib`
2. For your Web Browser, enter: `/usr/bin/mozilla`

This information is entered only once. When you log out, it is saved by the application in your home directory as `.drjava`. While you are on the preferences screen, take a few minutes to explore some of the other features that you can customize. If you don't understand them all, don't worry, the fog will lift as you progress through this course. When you have finished setting all of your preferences, select the `OK` button to return to the main drjava window.

Let us have a closer look at the main drjava window. It is partitioned into three main panes:

1. **File Listings Pane** displays a list of all files that are currently opened for editing in your session. It is located in the upper left-hand side of the window. To switch between files, simply double-click on the name of the file that you wish to display.

2. **Program or Text Pane** displays the text contained in the file that you have selected, or a new file which you are creating, in the upper right-hand side of the window. Let us add a short program and compile it. Perform the following:

   1. In the Program pane enter the following program:

      ```
      public class Greet
      {
        // This program prints a simple greeting
        public static void main(String[] args)
        {
          System.out.println("Well hello again!");
        }
      }
      ```

      Notice that as you type, the text colours may vary. Why do you think this happens? Modify your file by inserting and/or deleting blanks at the beginning of some of the lines in order to misalign your indentation. Then select the entire file (using `Ctrl + a`) and press the `Tab` key. This is a great way to improve the appearance of your program and make it more readable.

   2. Save the program by selecting the Save button. A window will appear showing that the file will be saved as a Java source file called `Greet`. This is what you want so ensure that the location of the file is where you want it, and select OK. Your file is now saved as `Greet.java` and you are ready to compile.

   3. To compile the program, select the Compile All button. The results of your compilation are displayed in the Compiler output tabbed section of the lower pane. We will look at this in our discussion of the interaction pane which follows.

3. **Interaction Pane** allows interaction between you, the programmer, and your program. Specifically, the interaction pane allows you to review the results of your compilations (in the Compiler output tab), allows your compiled program to accept and display data (in the Console tab), and allows you to interactively execute your programs and other Java statements (in the Interactions tab).
   Let us look at the Compiler output tab first. When you compile your program, providing you have entered the program correctly, the Compiler output section displays:
   `Last compilation completed successfully.`
   This means that your program has compiled cleanly and is ready to be executed. What would you see if there were a problem? Try creating a problem by removing the last closing brace from your `Greet` program. Remember to save the changed program, and then recompile it using Compile All. Now the Compiler output tab displays an error message as well as the line number on which the compiler thinks the error occurred. If you have selected Show all line numbers in the Display Options on the Preferences window, then you can more easily find the erroneous line in your program. Fix the program, save it, and recompile it.
   Once the program compiles correctly, it can be run. There are three ways to run a Java program in drjava. The first is to select Tools → Run document's Main method. The second is to enter F2. The third is to go to the Interactions tab and enter java Greet. Where is the output for the program displayed?
   Finally, let us examine some of the other uses of the Interactions tab. Try entering the Java expressions and statements shown below to see how drjava responds. To continue exploring, make up some entries of your own.

```
int myNumber = 3;
myNumber
double mass;
mass
char symbol;
byte age;
boolean passFail;
myNumber = 5.5;
myNumber = 5;
myNumber
symbol = 'a';
```

Transferring Files Between Computers: `scp`

When logged in to a computer on the ECF system you are using what is known as a *distributed file system*. This simply means that your files are visible to any ECF machine you happen to be logged in to, be it `p42.ecf` or `skule.ecf`. As long as you do all your work on ECF machines, you need not worry about having to transfer files.

However, for APS101, many of you may elect to work at home (or elsewhere), and will have to transfer your files to ECF so that they can be submitted for marking. To do this, you can use a utility named `scp`, which is a *secure* version of `cp`, and behaves in a very similar manner. To practice using `scp`, we will copy a file from whichever machine you are currently logged in to to `p1.ecf`. Try typing

```
$ scp Hello.java yourLogin@p1.ecf.utoronto.ca:Hello2.java
```

where `yourLogin` should be replaced with whatever your login name is. You will be prompted for your password. You may also be given a warning the first time you transfer a file to a new machine, that looks like this:

```
The authenticity of host 'p48.ecf (128.100.8.98)' can't be established.
RSA key fingerprint is 62:b8:e7:02:67:5c:1e:3b:de:e0:cd:cf:37:7d:7f:d7.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'p48.ecf,128.100.8.98' (RSA) to the list of known hosts.
```

It's OK to answer 'yes'.
Since we were copying the file to the same file system, this command has been demonstrated in a way that copies the file to a new name. Typing `ls -l` will show the new copy. If you wanted to copy the file to a machine on another system, say `seth.eecg`, without changing its name, you would type

```
scp Hello.java yourLogin@seth.eecg.utoronto.ca:
```

If you wanted to copy a file *from* that remote system,[1] you could type

```
scp yourLogin@seth.eecg.utoronto.ca:someFile.java .
```

The main difference between `scp` and `cp` is the addition of the construct `username@system:` at the beginning of the name of either the file being copied or the location the file is being copied to. The '`:`' character is important, as it tells `scp` where the pathname begins. When the first character after the '`:`' is not `/`, `scp` assumes that a pathname relative to the login name's home directory is being used. So, for example, the command

```
scp someFile maclean@p2.ecf:aps101/
```

copies the file `someFile` from the present working directory on the machine you are currently logged in to, to the `aps101` subdirectory of the home directory of user `maclean` on `p2.ecf`. The command

```
scp someFile maclean@p2.ecf:/tmp/
```

would transfer the file to the `/tmp` directory on p2.ecf—this is an example of using an absolute pathname.

In order to transfer files from a PC running windows, you will need to download a Windows version of `scp`. If you use Google™ to search for `WinSCP` or `PuTTY` you will find free programs for Windows that transfer files to and from your PC. In particular, the `PuTTY` download page has a utility `pscp` which behaves the same as `scp`. Note that these programs will not allow you to transfer files to and from your PC if you are located in the lab and not in front of your PC: this is beyond the scope of this lab.

Submitting your Assignments

You will submit your assignments electronically to be marked. To submit a program for marking, use the `submitaps101s` command as shown in the following example:

```
% submitaps101s 0 Hello.java
```

This command will submit the file `Hello.java` for lab 0. You can continue to work on your own version of `Hello.java` without affecting your submission. To submit code for other labs, replace the 0 with the lab number.

You can also submit more than one file, either by running the `submitaps101s` command multiple times, or by placing all the files on the command line:

```
% submitaps101s 0 Hello.java Goodbye.java
```

Note: If you choose to resubmit a file after making changes, the older version is replaced with the newer one. However, you can never erase files once they are

[1] In the following examples, the name `seth.eecg.utoronto.ca` is a real machine, but since you (almost certainly) do not have your own account on this machine, you cannot expect these examples to work if you try them.

submitted.

To view the list of files you have submitted for lab 0, use:

```
% submitaps101s -l 0
```

Note the `-l` is a dash-ell. This command will show you the files submitted, one per line. The columns show a variety of details, such as the time each was submitted and the size of the submitted file. Use this information to remind you which version you have submitted, since there is no way for you to read your submitted files.

Electronic Mail

There should be an easy to use GUI mailing application called `thunderbird` installed on ECF machines (try the `thunderbird` command).
UNIX itself has a very primitive mailing tool that is invoked using the `mail` command. To send mail using this facility, use the following steps:

1. Enter the command:
   `% mail <yourFriend'sEmailAddress>`
   If their account is also on the ECF system, you can drop the `@ecf.utoronto.ca` part of their address.

2. The `mail` command will prompt you for the subject of the message. Type in a short description of the subject and then hit RETURN.

3. Type in the message.

4. After finishing the message, hit RETURN. Type in a single period and hit RETURN again and the letter will be sent.

If at any time while you are composing the letter you decide to cancel it, enter CTRL-c twice; that is, hold down the CONTROL or CTRL key (as you would the shift key) while typing in a c.

If someone has sent mail to you, UNIX will notify you with the message `You have new mail` when you login. To read new mail, enter the command:

```
% mail
```

The UNIX mail facility will display a list of the messages waiting to be read. For each message in the list, the sender's login and the subject will be displayed. To read the messages all you need to use is the Enter key and the space bar. Press Enter to display the first letter in the list. If it cannot completely fit on the screen, you will see only one screenful and a `more` prompt; the space bar will show you the next screenful. When that letter is completed, you can press Enter again to read the next one. This is the most basic way to use e-mail, but more control over the program is possible. For more information, see the UNIX manual.

There are better mailing alternatives that may take a little bit of learning. Try the Pine mailer. You can learn about this mailer using the `man pine` command and also by asking the program itself for help, using its internal help commands.

If you wish to have mail that is sent to your ECF account sent to at another address, you can forward them to the other email address (for instructions see `https://postbox.ecf.utoronto.ca/`).

Browsing the World Wide Web

You should be able to use the Mozilla button (the icon depicting planet earth encircled by a mouse and its cord) on the taskbar to start Firefox, a web browser like Netscape or Internet Explorer. You can also start Firefox by typing, in a window:

```
% firefox &
```

After a delay, you may see a window asking you if you will accept various terms and conditions. Read it and then click the OK box if you wish to proceed. You should note that there may be different versions of browsers on the system.

From Firefox, you can gain access to the course website:
`http://portal.utoronto.ca/`

You may also find it worthwhile to check out the Engineering Computing Facility site:
`http://www.ecf.utoronto.ca`
which has all sorts of information about ECF facilities.

Setting Up Your Own Web Space

You can begin creating your own web space on ECF right away. At the UNIX prompt, type:

```
% cd ~
% wwwinit
```

A series of instructions will appear on the screen telling you that the space was created. You can visit your home page using any web browser by replacing login with your login id:

`http://www.ecf.utoronto.ca/~yourLogin`

Logging Out

To log out, click on the red hat button on the taskbar. This will produce a pop-up menu with a number of options. Click on Logout. In the window that pops up, confirm that you want to log out by clicking OK.

Never forget to log off when you are done! If you do not log out, someone will be able to get unauthorized access to your programs and other files after you leave the terminal. They may even leave "back doors" open to regain access to your account after your password is changed. They could get you in BIG trouble, so always ask a sysadmin (system administrator) for help if you think your account is no longer secure.