University of Toronto
APS105H1F — Computer Fundamentals
Lab 0: UNIX Basics

Introduction

This first lab session will serve as your introduction to the computers we'll be using in this course. They are running a variation of UNIX called Linux, specifically the version compiled by Red Hat. UNIX is significantly different from the Windows or Macintosh environment that you may have used previously.

This assignment is in the form of a tutorial that will gradually take you through the basics of interacting with the computer. You may find that this assignment will take longer than two hours to complete. If this is the case, please complete it at another time, but as soon as possible. This lab will not be marked

After you complete this assignment, you are encouraged to continue your introduction to the laboratory computing environment by reading the manual *Getting Started With UNIX*. It can be found online at

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

Logging into the Computer System

The first thing you must do when you begin your work on the computer is to "log in" to let the system know that it is you who is using the system. Access to the system is controlled to maintain its security; only those with accounts on the Engineering Computing Facility (ECF) system can access it. To login, you must have two pieces of information: a login name and a password. All undergraduate electrical and computer engineering students are automatically given an account when they register. If you do not already have a login name and password, you will be told how to obtain them in the next section.

Your login name (or "login") is a six to eight character string that is your identity on the computer network. Your password is a variable length character string that should be known only to you. Your password serves as a key to your account much as you would use a key to gain entry to your locked house. In fact, not even the system knows your password; it only keeps an encrypted version. Later in the lab you will be shown how to change your password.

Never give your password to someone else or let someone use your account. Doing so is against University policy and the other user can leave open a "back door" into your account to access it later, even if you change your password.

Just as you login to the computer when you begin to use it, you must logout of the computer when you are finished using it. If you must logout before you reach the end of the assignment, read the final section of the assignment which instructs you on how to logout. After you logout, DO NOT TURN THE COMPUTERS OFF!

Getting Your Login Name

This is how you get your account name if you have never logged in before. If this doesn't work for you, go to the ECF office. It is located in the Engineering Annex, in Room EA206.

1. At any ECF terminal in SF1012 or SF1013, you should see a screen with a window that asks for a username (see p1 of lab1Screens.pdf). In the `Username` field, type `getname` (all lower case).

2. You will then be presented with a window that will ask you to enter your 10-digit student number, your last name, and your date of birth (see p2 of lab1Screens.pdf).

3. Shown below is an example of what you will see on the screen. The text you enter should replace the student data shown here with your actual student number, name, and birth date. In this example, the initial password would be 43210108. You'll learn how to change it later on in this lab.

```
Please enter your student number: 0978654321
Please enter your last name: Smith
Please enter your date of birth as YYMMDD: 990108

Your ECF login name is: smithjo
```

4. If the information entered is correct and you are registered, the program will display your ECF userid and ECFPC userid. You will use the ECF userid for this course.

5. Your initial password will be eight digits long. It will consist of the last four digits of your student number followed by the month and day of your birth in the form MMDD.

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). Linux will then start up the *X Window computing environment*. You will see as you progress in this course that the X Window environment of the UNIX operating system is a powerful one to work in because of its ability to have the computer work on a number of different things at once, a concept known as multitasking. You can even run programs on different computers and have their windows appear on your screen. The X Window environment also allows you to conveniently organize the windows on the screen. For now, we will learn the basics about manipulating windows.

When the X Window system starts up, there will be a *taskbar* along the bottom of the screen (see p3 of lab1Screens.pdf). There may also be windows on the screen or you may have to create them yourself. When you log out, the desktop is saved so that the windows and their positions are saved for your next login session.

Most of your work will be done using *terminal windows*. Start a new terminal window by right-clicking some unoccupied portion of the screen, and select `New Terminal` from the menu that pops up.[1] A terminal window will pop up. The window can be moved around on the screen with the mouse by by clicking and holding the left mouse button over 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 lower right corner of the window. You should see the cursor change to an L-shape that aligns with the corner of the window. To make the window bigger, press the mouse button, drag the cursor away from the window, 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 makes it shrink to an icon on the taskbar. To make the window reappear, click on its name in the task bar.

In the terminal window, you should see some introductory messages about the system followed by 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

[1] You can create an icon in your task bar for this very useful terminal window by right clicking the empty middle part of the task bar (called the *pane*), and navigating through the menus as follows: **Add to Panel → Launcher From Menu → System Tools → terminal** — this process is shown on p5 of lab1Screens.pdf.

the mouse so that it is on top of the Login Session window and enter the following command:

```
$ xterm &
```

The `xterm` command stands for X Terminal and it creates a window that, like the GNOME terminal window, has a UNIX prompt at which you can enter commands. You can now use either the GNOME 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. Despite having many windows on the screen, only one window is active at a time. The active window has a highlighted border, 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 (or a numeral) of each word. For example, the sentence "I hate to be late for computer cla ss" 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>
```

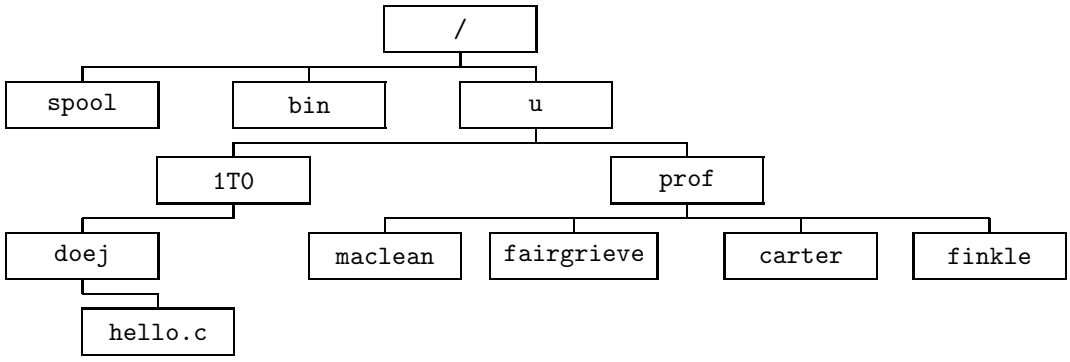
As you type both the old and new passwords, the characters will not be visible on the screen. This is deliberate, to prevent others from seeing your password by looking over your shoulder. Because you cannot see your password, you are asked to type it twice, on the theory that you are unlikely to make 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, other files contain text that can be read, while other files contain images that can be displayed on the screen. Two things that all files have in common are that they can be stored, 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. 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 the 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, type in the following command:

```
$ pwd
```

UNIX will respond with a string that looks like `/u/1T0/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 `1T0`, 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` 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/1T0/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 do a `pwd` command, you will see that you are now in `/u` even though you didn't 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 `1T0` directory by entering:

```
$ cd 1T0
```

Again, if you enter the `pwd` command you will see how 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. But 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" and instructs UNIX to list the contents of the present working directory, which will include any files and subdirectories. Now let's 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 want to explore some of these files and determine what they do some other time.

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

```
$ ls -al
```

The final `l` stands for long form and 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, it was 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 aps105
```

The argument supplied with this command is the name of the new directory, `aps105`. 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 other 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
```

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 C++ program `hello.c` which is located in the directory `/share/copy/aps105` to your home directory:

```
$ cp /share/copy/aps105/hello.c .
```

The first argument of the `cp` command is the source file; here it contains the absolute path of the `hello.c` 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.c` (located in `/share/copy/aps105`) in the current directory and give it the same name: `hello.c`. Now try using the `mv` command to place this file in the `aps105` directory you created earlier:

```
$ mv hello.c aps105/hello.c
```

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.

Creating, Compiling and Running a C++ Program

There two approaches to creating, compiling and running the C++ programs that you will write for this course: Either you can use a separate tool to perform each of these actions – a text editor to create the program file, a compile command to compile it, and another command to run it; Or, you can use an Integrated Development Environment (called IDE), which allows you to perform many tasks from a single, usually graphical program interface. In this course we will only use the first approach.

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 C++ 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 it is described below. The editor you should learn to use for your assignments is called *nedit*, and you should try using it fairly soon. Call up nedit on your computer using the command:

```
$ nedit &
```

The nedit window that pops up differs slightly from the xterm windows we have been using. This window is specifically designed for editing text files. There are two areas we need to understand in this window to this window. Along the top of the window is a menubar with categories of actions that may be performed. The bottom section is a large empty box and it is here where the text file is composed. nedit starts up with this section cleared so you can create a file from scratch. Let's create a simple text file. Move the mouse so that the cursor is pointing in this section. Then write some simple text in this box such as your name or "testing testing 1 2 3 . . . "

Now lets save our file. Because this is a new file, it has yet to be named. To save our new file select `File` → `SaveAs` from the Menubar. Another pop-up box will be displayed which allows us to identify the new filename and where (in which directory) we wish to store this file. Point the cursor inside the box labelled *Save File As* and append `test.txt` to the path which is displayed, then click on the `OK` button. 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. Besides creating new files, nedit can be used to change existing files. To open an existing file, `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 first `testing` word and click. 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 and your changes will be saved. 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` allows you to look for words, or replace them in your text. `Preferences` will allow you to highlight the structure (syntax) of your C++ programs and show your line numbers (useful for finding problems). `Shell` will allow you to execute commands to compile your C++ 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 you may want to try out are `pico`, which is also for beginners, and `vi` or `emacs` which are for more advanced users.

Editors are useful for creating and editing text files, but if you just want to read the contents of a text file, you can use one of several UNIX commands at an xterm window. This saves you from having to start up a text editor to view the file. The most convenient command for typing out a text file is the `more` command, but most UNIX systems have a better version of `more`, called `less`. Entering `more` (or `less`) along with the text file name as an argument types out the file in the xterm window, (one screen at a time, if the file is longer than one screen). You can advance through the file by pressing the space bar. Try using the `more` command as follows:

```
$ more test.txt
```

Compiling and Running a C++ Program

Now try to compile the `hello.c` program. From a terminal window, go to the aps105 directory and start the C++ compiler by typing:

```
$ g++ hello.c
```

This will create a file named `a.out` in the aps105 directory. This file contains the machine code that will perform the desired results when run on the computer used to compile it. You run this program as follows:

```
$ ./a.out
```

You should see the output of the program, saying `Hello, world!`. If you prefer for your program to have a more interesting name than `a.out`, try

```
$ g++ hello.c -o hello
```

which directs the compiler to place the output in the file named `hello`, which can be run with the command

```
$ ./hello
```

What happens if you instead try to run your program with the following command?

```
$ hello
```

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 APS105, 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.c yourLogin@p1.ecf.toronto.edu:hello2.c
```

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 hos
ts.
```

It's OK (for this lab at least) 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.c yourLogin@seth.eecg.utoronto.ca:
```

If you wanted to copy a file *from* that remote system,[2] you could type

[2] In the following examples, the name `seth.eecg.utoronto.edu` 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.

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

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 apathname relative to the login name's **home** directory is being used. So, for example, the command

```
scp someFile maclean@p2.ecf:aps105/
```

copies the file `someFile` from the present working directory on the machine you are currently logged in to, to the `aps105` 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^{TM}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.

Electronic Mail

UNIX 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 can't 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 asking the program itself for help, using its internal help commands.

If you wish to have mail that is sent to your ECF account forwarded to you at another address, you can create a file called `.forward` whose contents are simply the address (or addresses) to which you want mail forwarded.

Important: You are required to read e-mail sent to you at your ecf e-mail address, as it may contain important course information (*e.g.*, your assignment marks for this course). *We are not responsible for e-mail that goes missing if you forward to another e-mail address.*

Browsing the World Wide Web

You should be able to use the FireFox button (the icon depicting planet earth encircled by a mouse and its cord) on the taskbar to start FireFox, a Netscape-like browser. If not, you can 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 Netscape on the system.

From Netscape, you can gain access to the course website:
`http://courses.ece.utoronto.ca/20069/aps105h1f/`

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 on skule, 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 Netscape by replacing login with your login id:

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

Logging Out

To log out, click on the GNOME footprint button on the taskbar. This will produce a pop-up menu with a number of options. Click on `Log out`. Then confirm `Really log out?` by clicking `Yes`.

Never forget to log off when you're 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 for help if you think your account is no longer secure.