Addendum # 0

Unix for EE 201L-ers

By

Bilal Zafar and Gandhi Puvvada

Acknowledgement: Most of the material covered in this addendum is taken from user guides prepared by the Information Services Division at the University of Southern California. Please visit http://www.usc.edu/isd/ for more details on the topic covered.

Part 1: UNIX Concepts

Solaris vs. Unix vs. Linux

UNIX was developed by AT&T Bell Labs in 1969. Later, a research group at University of California at Berkeley added some very important modules to the basic "kernel" of UNIX. This version of Unix is usually referred to as the **Berkeley Unix**. It was here that Unix really took off as the operating system of choice for developers who wanted to add functionality to the kernel so as to fit their needs. Several companies took the Unix kernel and added various features to it to suite their machines and target market. **Solaris** is the operating system distributed by Sun Microsystems which uses the Unix kernel. The differences between Unix and Linux, however, are more subtle and therefore, arguable. Google the phrase "differences between Unix and Linux" if you really care to find out. From a novice user's standpoint, there are really no differences. Almost all basic Unix commands work on Linux.

USC's Computing Network

The portion of the computing resources of ISD (Information Services Division) that we are interested in are typically referred to as "student computing facility" (**SCF**). We have the two main servers "**aludra**" and "**nunki**" which are time-sharing machines which means that multiple users can work on them simultaneously. Obviously, they are both quite powerful machines (4 or 8 processors, gigabytes of memory, etc.) but one particular user gets only a small fraction of these resources. Interestingly, although aludra and nunki are different machines, from an user's standpoint they are identical, that is, they have the same view of the file system (if you create a file on aludra, you can immediately see it on nunki) and have the same applications available on them. The SUN workstations available in the user rooms (**SAL**, **LVL** and **KOH**) also share the file system with aludra and nunki.

Common Desktop Environment (CDE)

Common Desktop CDE (Common Desktop Environment) is one of the several "front-ends" of Solaris. It provides a friendlier user-interface than the classical Unix command line interface. You can think of CDE as a nice wrapping on top of Solaris – just to make you feel like you're working on Windows/MacOS. While working on SUN workstations, we will use CDE (not 'Open Windows') as our default interface. In fact, Open Windows won't allow us to run certain applications that we need for EE201L.

Exercise 1: Ready-Set-Go

- Is this your first time logging on to a SUN workstation here at USC?
- Using the user name and password that you use to check your USC email, log on to your workstation. How many windows appeared on your screen when you logged on? Can you identify them?
- Identify the following buttons at the bottom of your screen:

1 File Manager
2 Terminal
3 Exit
4 Workspace switch

- If a terminal window did not open automatically when you logged in, click on the terminal button to open a new terminal. If you already have it open, you don't need to open another one. What does your prompt look like?
- Type in the Unix command "date" at the prompt and hit enter. What is the date and time reported by the system?
- Type "exit" to close the terminal.

Please refer to the end of this addendum to get to know hoe to login from a home PC.
http://www-scf.usc.edu/~eeview/README.vnc.setup.pdf

Unix Shells

The UNIX shell is a program that accepts commands from a user and executes them. Several UNIX shells are available here at USC including:

csh C Shell
tcsh T Shell
bash Bourne-Again Shell

Exercise 2: What's in the shell?

Let's check our current shell setting.

- Open a new terminal window.
- To find which shell you are currently running, type:
  echo `$shell`
  If the system responded with `/bin/tcsh` (or `/bin/csh`), it means that you are running the T shell (or C shell). You don't have to change it. If you are running bash, we suggest that you change your default shell to C shell or T shell permanently. Seek help from your TA.
- To change you default shell (permanently), type **chsh** and enter your password when prompted. Enter `/bin/tcsh` as your new shell.
  If you just changed your shell, close the terminal by typing "exit", open a new terminal and check your current shell (by typing echo $shell).

Student accounts are initially configured to use the C shell. For our purpose, you may use either C shell or T shell (you might find T shell to be a bit more convenient though). However, you SHOULD NOT use bash

(even though some students find it 'fancy') because ISD doesn't support it and many of the applications available through SCF are not configured to run under bash.

More Information: There are some fun things you can do with your shell settings. For example, you can change your default prompt. To read more about shells, visit ISD's user guide on Unix shells at:
http://www.usc.edu/isd/doc/os/unix/concepts/shell.html

Unix Commands and Syntax fundamentals

We just saw a Unix command "echo" and a utility "chsh" but we will need more Unix commands to be able to create, copy, move and delete files and directories (folders) etc.
A unix command usually has up to three components:

```
command -[options] <arguments>.
```

"Command" refers to the actual keyword which is interpreted by the shell, "options" control the behavior of the command while "arguments" are entities the command is applied on (file or directories, usually). The syntax rules require that (a) the command, options and arguments must be separated by spaces and (b) a minus (or hyphen) sign should precede the options. Let us try to understand this with the help of a popular command: list.

We will look at several other unix commands and utilities later in this lab.

Exercise: The list command

List (ls) is perhaps the most commonly used Unix command. It lists the contents of a directory. Let's give it a try.

- Type at the prompt:

ls

  Notice that we did not use any options or arguments and the system returned a simple list of all the files and folders in the *current* directory.

- Now, let us tweak this command with some options. Type at the prompt:

ls –a

  Is the list of directory contents returned by "ls –a" longer than the one returned by just "ls"? Did you notice some files starting with "." reported by "ls –a"? This is because the option –a forces the system to return ALL files, including system files and hidden files, to be displayed.

- Now, let's try another flavor of the same command. Type

ls -la

  What additional information is produced by "ls –l"?

- We can specify more than one option for most unix commands. For example, the option "-t" sorts the list by "time". Let us combine this with the "-l" option. Type:

ls –lt

  Notice the dates corresponding to the files and directories reported.

- We can also give a folder as argument to the list command to view the contents of *that* directory rather than the current directory. Look for a directory in your home directory and use the command syntax

File System Structure

It is important to understand how directories are arranged in Unix, especially with this shared file system view between aludra, nunki and the SUN workstations here at USC. Logically, Unix files are organized in a tree-like structure (just like in Windows). The top level directory is called root and is represented by "/" (which is the same as C:\ in our typical Windows systems). Under the root are several directories. Some of the more important ones are:

home usr var tmp

Everyone's home directory is located under home. Your home directory is where you are when you log into the system. Most Unix applications are located under usr. tmp is a shared directory where you can temporarily store your files (for example, if you have a 10MB file that you want to send to your friend. You can simply put it in tmp and ask him to copy it from there).

Exercise: Lost in the directories?

- To see which directory you currently are in, simple type the print working directory command:

pwd

  What is your current location?

- Change directory command **cd** is used to navigate between directories. Change directory to any other sub-directory in your home directory (mail, for example) by entering the command:

cd <directory_name>

- To return to the home directory from ANY directory, just type "**cd**" (without any options or arguments)
- To go to the home directory type "**cd ~**".
- To go to tmp directory (from your home directory), type: **cd /tmp**
- In Unix commands, the current directory is referenced by ".", the directory one level above the current is referenced by ".." whereas the home directory is referenced by "**~**".
  For example, suppose you were in the directory `home/ee201l/homeworks`
  and you wanted to go to the "ee201l" directory you would type: **cd ..**
  but if you wanted to go back to the home directory directly, you would type: **cd ~**

Login Information

There are two files in your accounts "**.login**" and "**.cshrc**" which contain initialization information for your account. When you login to your account, your `.cshrc` file is executed (or "sourced", in Unix terminology) first, followed by your `.login` file. The `.cshrc` file determines the architecture and operating system of the host you have logged in to and defines some shell variables appropriately. The .login file defines some environment variables needed to run certain applications.

When you set up your account for ePD, you will notice that a few lines will be added to your `.login` file. These lines actually source a setup file in our account "`eeview`" which will allow you to run the application. By putting these lines in the .login file, we make sure that your account is ready to run ePD every time you log in.

Please note that you should always save your old .login or `.cshrc` files before making any changes to it. One way to do it is by saving the original file as something like `OLD.login`. and then make the changes to the `.login` file. If your `.login` or `.cshrc` ever gets corrupted, you will notice your prompt will be changed to '>' or '$'. You can copy a fresh copy of the initial file by doing:

```
cp /usr/usc/skel/.cshrc .
cp /usr/usc/skel/.login .
```

Of course, you will have to re-do the setups for all the applications that wrote something to your `.login` or `.cshrc` files.

If you forget your password, contact ISD at (213) 740-5555 to have it reset. You can change your password by typing in “passwd” command.

Part 2: UNIX Commands - I

Recall that the general syntax of Unix commands is:

command –[options] <argument>

In this section, we will introduce some basic Unix commands along with some of the options that are available for each of them.

Directory Management

Command	Example	Description
mkdir	mkdir test	Creates a directory called 'test' under the current directory
cd	cd test	Change to directory named test from the current directory (test should be a sub-directory of the current directory)
pwd	pwd	Show current working directory
ls	ls **-**l	List contents of current directory
cd **..**	cd **..**	Change to directory one above the current directory
clear	clear	Clears your terminal screen if possible
rmdir	rmdir test	Removes an empty directory called test

File management

Command	Example	Description
cp	cp /tmp/ee201.txt **.**	Copy the file ee201.txt from /tmp to the current directory. Notice the period (**.**) at the end of the command which means that the destination is the current directory.
mv	mv ee201.txt ~/test/ **.**	Move ee201.txt to directory called test.
rm	rm ~/test/ee201.txt	Remove (delete) the file 201.txt in the directory ~/test/201.txt
chmod	chmod +x ee201.txt	Make the file ee201.txt an executable file
/bin/rm –R	/bin/rm –R ~/test	*Recursively* remove all files and sub-directories inside the directory ~/test including the directory ~/test

Part 3: Text Editor in UNIX

Most Unix systems provide several test editors. **vi** (short for “visual editor”), **pico**, and **emacs** are the three most common editors. emacs is perhaps the most user-friendly of these three editors, therefore, we will only discuss emacs here. However, emacs does not work well over a PuTTy connection. There will be no graphical interface and pull down menus if you run emacs in a PuTTy terminal. So we encourage you to learn pico as pico is easy to learn though not very powerful.

emacs

In emacs, you can open multiple files simultaneously and switch between them with ease. Each of these files is opened in a separate “buffer” and you can switch between buffers.

Exercise: Let’s “emacs” a new file
To open or create a new file, simply type “emacs <file_name> &” at the prompt. “&” at the end of the command is optional. It means “run the command in background”. *This is possible if you are logged on a SUN machine or running vncserver on your PC. It is not possible in a PuTTy terminal.* On a SUN, however, it is recommended that you use the “&”. If you do not put the “&” sign at the end of the command, the prompt (i.e. the window) will be committed to the emacs process only and will not be available to you for entering any other commands until you close emacs.

Type “emacs welcome.txt &” at the prompt.
emacs will open. Enter whatever text you want.
Save the file by selecting the option from the pull down menu (File -> Save Buffer) or using the hot-keys <CTRL>+X <CTRL>+S. You will notice “**” appear at the bottom of the screen if your current buffer is not saved. Exit emacs by selecting the option from the pull down menu (File -> Exit Emacs) or using the hot key <CTRL>+X <CTRL>+C.
Like most other text editors, Emacs auto-saves your work periodically. These auto-saved files have names of format “#<file_name>#”. Also, when you modify and save a buffer, the unmodified (previous) version of the buffer is saved with the name “<filename>~”.

There is a lot more you can do in emacs, however we will leave it for you to discover all those features.

Part 4: Unix Commands - II

In this section, we will see some more advanced unix commands and utilities. Utilities are small packages that work just like commands but perform operation more complicated than a normal Unix command.

grep

grep command searches an entire file (or a set of files) for the pattern you want and displays its findings. Interestingly, you can use this command to search for a pattern in the results of another command. For example, if you were looking for all the files whose name includes the keywords “ee201” and “homework” in any order, you can combine the grep command with the ls command as shown below.

```
ls -l | grep ee201 | grep homework
```

Notice the symbol "|" (called "pipe") between the two commands ("ls" and "grep"). Pipe is used to forward the output of the *left-side* command as input to the *right-side* command. You can use the grep command to print all the instances of a particular keyword in a file.

```
grep <key_word> <file_name(s)>
```

more

The more utility displays the contents of a text file on the screen, one screen-full at a time. It is a handy tool to see a file without opening a text editor. For example, you can see the file you created "welcome.txt" using the more utility by typing:

```
more welcome.txt
```

After seeing one screen-full of the file, to see the next screen-full, please press the space-bar on your keyboard once. Instead of the pressing the space-bar, you if press the enter key, one more line of the file is displayed.

You can also concatenate the more utility with commands like "ls" to display only one full screen at a time (in case you have too many files/directories). The syntax for that would be:

```
ls –l | more
```

tar

tar is a powerful utility to create an archive or extract files from one. It is similar to the winzip on PCs which allows you to create a zip file and later extract files from the zip file. The *tar* command is useful if you wish to backup a large number of files and/or transfer them. Without detailed explanation of each option, we will just give the commands for the modes in which tar utility can be used:

If you want to tar a directory, the command will be:

```
tar -cvf <file.tar> <dir_name>
```

Where file.tar is the name you want to give your tar file, and dir_name is the name of the directory you want to tar.

To list in detail the contents of a tar file, use the following command:

```
tar -tvf <file.tar>
```

If you want to extract files from a tar file, use a command similar to the following:

```
tar -xvf <file.tar> <filenames>
```

where file.tar is the name of the tar file from which you would like to extract files, and filenames is a list of files you would like to extract. If you do not specify any file names, then tar extracts them all.

Note: tar extracts files to your current working directory with the same names that the files had when they were archived. Be careful not to overwrite already existing files with the same names.

echo

echo utility outputs its argument to the screen. We can read the values of certain "environment variables" using this command. Type:

```
echo $DISPLAY
```

The system will return the value to which the DISPLAY variable is set to.

```
echo $PATH
```

The system will return the value to which the PATH variable is set to.

which

"which" locates a command; displays its pathname or alias. For example, type:

```
which mkdir
```

env

the command "env" lists all the environment variables like the path, the display setting etc. just type "env" and you'll see a long list of variables and their values. If you are interested in looking at the value of a particular variable, you can always concatenate "grep" command with "env". For example, if you want to see the value of the variable "DISPLAY", you would type in the command:

```
env | grep DISPLAY
```

Listing and killing running processes

"ps" command lists all the processes currently running. Since you can have multiple sessions running on aludra and nunki simultaneously, and just "ps" gives you a list of processes running under the current session, we recommend the following switches:

```
ps –aux | grep <username>
```

Basically, ps –aux gives a list of ALL the processes running on the machines. Since many users are logged on and you really don't care about the processes launched by other users, you "grep" that processes owned by you out of that very long list. Here's a sample output of this command:

```
aludra-eeview{2} ps aux | grep eeview
eeview  1319 0.1  0.0 1576 1424 pts/212  S 18:27:39  0:00 -csh
eeview  1460 0.0  0.0  992  784 pts/212  S 18:28:19  0:00 grep eeview

USER     PID %CPU %MEM   SZ  RSS TT        S    START  TIME COMMAND
```

The numbers in the *second column* are the **process IDs**. We will use the process ID of a process to kill it.

Unix also has its low points! Yes, at times you might find an application (like ePD) crashing on you. However, you can kill the process very efficiently (unlike in certain other operating systems). As we mentioned before, we use the process ID to kill a process. The command is:

```
kill <process ID>
```

At times a simple kill signal will not work. You can then use –9 option to kill the process along with its children (child processes):

```
kill -9 <processID>
```

If you want to kill all the processes that you are currently running, the command is:

```
kill -9 -1
```

Note that this will kill ALL the running processes owned by you, INCLUDING the terminal that you are using to type this command. Do not use it as it could hung up your terminal window.

Putting a process in bg, suspending processes and bringing to fg

As mentioned before, putting the "**&**" sign at the end of an invocation makes the program run in the ***background*** (and therefore, not commit a terminal to itself). You can ***suspend*** a process while it is running (after it has been launched) by using **CTRL+Z** when the processes you wish to be suspended is selected. Note that suspending a process is different from running it in background. A suspended process won't

"behave normally". You will have to bring the process to the ***foreground*** by giving the command "**fg**" to start working on it or let it work in the ***background*** by issuing the command "**bg**".

Quota

Every student has 100MB of disk space on the SCF file system. Your applications might not run (properly) if you run out of disk space. In fact, your account may be completely blocked by ISD if your account stays over your quota limit for some days. We strongly recommend you monitor your quota regularly (say, every couple of weeks), just to make sure you are well within your limit. To check how much disk space you have left, use the command:

```
quota –v <username>
or
quota –v
```

It basically gives you a list of disk spaces you have been allotted on all the disks of SCF file system. You will have your account setup on one of these disks only so concentrate on that line in the result.

To see how much space is being taken by each directory in your account use the command "**du | more**" or "**du –a | more**". Note that the **du** command gives you a list of ALL the directories in your account and the **du -a** command gives you a list of ALL files in ALL the directories (including the directories) in your account. So it can be a pretty long list. Use it only if you are having a hard time figuring out where the big files are that you should remove to free up more space.

One of the things you want to look for is to see if the system dumped a core file in your account when ePD or any application crashed. The core files are usually big (several megabytes) and cause your account to exceed the quota allocated. To look for core files, you can use the **find** command as follows:

```
find ~ -name core
```

Remove the core file using the **rm** command.

Printing

Unfortunately, printing on Unix is not as straight-forward as it is on Windows/Mac machines. You can directly print post-script files using the "lpr" command. The syntax for that is

```
lpr -P<printername> <filename>
```

Example:

```
lpr -Pps_ucc101 ee201l_lab3_sch.ps
```

The printers available in the various user rooms are listed below;
ps_lvlg40
ps_lvlg40_dp
ps_sal125
ps_sal125_dp
ps_wphb34
ps_wphb34_dp
The suffix "_dp" stands for "duplex" which means that the printer will print on both sides of the page.
To print text files (pure ascii files with .txt extension, not pdf or doc files) you can use the enscript command or the lpr command as shown below:

```
enscript -P<printer> [options] <filename>
```

Example:

```
enscript -Pps_sal125 lab.txt
```

Viewing a PostScript File

Use the utility "**ghostview**" or the command "**gs**" to view a postscript file. Note that postscript files are not editable. These are essentially files converted into the language of the laser printers (usually called 'post-script' printers).

```
ghostview ee201l_lab3_sch.ps
gs ee201l_lab3_sch.ps
```

Part 5: Logging on to an UNIX machine from a PC using VNC

Please refer to http://www-scf.usc.edu/~eeview/README.vnc.setup.pdf

Part 6: ePD and problems associated with installing and running ePD

Please refer to http://www-classes.usc.edu/engr/ee-s/457/ePD_problems.html .
Besides solutions to several common problems, it lists procedures to print schematics, waveforms.
Please bookmark this page on your home PC and refer to this as and when you face any problem with ePD.
Do not hesitate to contact your TA or
Prof. Gandhi Puvvada (**gandhi@usc.edu** EEB238 213-740-4461(O) 310-839-3933(H)) for help.