CS 215 - Fundamentals of Programming II
Fall 2012 - Very Basic UNIX

This handout very briefly describes how to use Unix and how to use the Linux server and client machines in the CS (Project) Lab (KC-265) and the Computer Systems Lab (KC-267). For more information, ask a CS Lab worker, look at the links on the CS 215 course page, or buy a book.

CS Lab (KC-265) and Computer Systems Lab (KC-267)

CS Lab & KC-267 consist of client machines that dual-boot Windows 7 and Ubuntu Linux. On boot up, the machines will give a menu to choose an operating system. Please reboot the machines properly if you need to switch to the other operating system. There is a Linux server machine named **`csserver.evansville.edu`**, that provides login and home directory service for the Linux sides of the client machines. Users with accounts on csserver also may map their Linux home directory as a network drive on any Windows machine inside the UE firewall.

Both labs are open 24/7 to authorized students with access provided by a cipher lock. Authorized students are those in current CS courses with need to access the labs and students working on projects for the CS program. A list of authorized students is sent to Security at the beginning of each semester. The combination to the locks are available from instructors and labworkers During each semester, there are labworkers with schedules posted outside the CS Lab who can answer questions regarding how to do things in the lab.

Students are allowed to log into csserver remotely using any terminal client supporting the SSH (secure shell) protocol, such as ssh (Linux) or putty (Windows). While you can log into csserver from a lab client machine using a terminal client, we prefer that you use the Linux side directly (unless a program must be run on csserver specifically) in order to keep the load low on csserver. More information on remote login is given at the end of this handout.

Basic UNIX functionality

UNIX is a command-line oriented operating system. This means that you interact with a computer through a interactive shell and programs are run by typing commands and their arguments at the shell prompt. (It is similar to DOS in this way.) The basic syntax of running any program is:

```
command options arg1 arg2 arg3 ...
```

where **`command`** is the name of the program and the options and arguments are whatever else it needs to run. Options are things that change the way a program behaves, typically they start with **`-`** ("dash") and may have arguments of their own (e.g. a file name). Arguments are things that the program will operate on, typically file and directory (folder) names. Note that UNIX is case-sensitive. Conventionally, most UNIX commands and file/directory names are in all lower case.

The UNIX file system is hierarchical. The root directory/folder of the file system is called **`/`** ("slash"). Under the root are various subdirectories. Each user has a home directory under **`/home`**, e.g. **`/home/droberts`**. When you log in, you are put into your home directory where only you have access to read and write files initially. The default shell on the CS Lab Linux server is **`bash`**. This shell supports, among other things, using the up and down arrow keys to access previously typed commands, line editing using emacs style commands, and command completion using the TAB key. (The shell can be changed using the command **`chsh`**, if another is preferred.)

Files and directories are named using either absolute or relative pathnames. An absolute pathname starts with `/`, the root directory. A relative pathname is relative to the current directory and can start with a subdirectory name, `..` ("dot dot" - parent directory), `.` ("dot" - current directory) or `~` ("tilde" - home directory). Most commands also will take wild cards in pathnames. The most common ones used are: `*` (match 0 or more of any character) and `?` (match exactly one character). For example, `project1.*` will match all files starting with `project1.` and having any extension (e.g., `project1.cpp`, `project1.o`, etc.). Likewise, `project?.cpp` will match any `.cpp` file that has the name `project` and one character (e.g., `project1.cpp`, `project2.cpp`, etc.).

Basic UNIX commands

Here is a very brief list of the most commonly used UNIX commands (more or less in the order presented in class) with the most commonly used options and arguments that you will need to know to get started. Unless otherwise noted, all commands may be used on csserver or the client machines.

`passwd`

Change password. The program will ask you for your current password, a new password, and then your new password again to confirm.

`smbpasswd`

Change Samba password (used only on csserver). Samba is a service that (among other things) exports a UNIX file system as a Windows file system. This password may be the same or different than the UNIX password and is the one used when mapping the Linux home directory as a network drive on Windows.

`mkdir subdir1 subdir2 ...`

Create subdirectories in the current directory.

`ls options directory`

List `directory`. Default is in alphabetical order ignoring hidden files (those starting with a dot). Options include `-l` (long listing format showing permission modes and sizes), `-a` (including hidden dot files), `-d` (directory only). Options can be combined, e.g., `ls -la` will list all files, including hidden files, in long format.

`chmod newmode name1 name2 ...`

Change permission mode. Permissions to access files and directories are granted in three groups (owner, group, world) of three rights (read, write, execute). They are represented in a long listing (`ls -l`) as:

```
        rwxrwxrwx
```

These permissions can be represented as a 3-digit octal (base 8) number where each right is a bit in the number. In the long listing, a letter (`r`, `w`, `x`) means the bit is set to 1, whereas `-` (minus) means the bit is 0. E.g., 777 would be all rights to all users (`rwxrwxrwx`), whereas 754 would be all access for the owner, read and execute access for the group, and read-only access for everyone else (`rwxr-xr--`).

By default, your home directory is open for all access to you and closed to all others, i.e., mode 700 (`rwx------`). All other directories and files are created with all access to you and read-only access for everyone, i.e., modes 755 (`rwxr-xr-x`) and 644 (`rw-r--r--`) for directories and files, respectively. You might want to change the permission mode of your course directories to 700 to prevent anyone from viewing your course files.

`man command`
Show the manual page for **`command`**. Most commands have man pages. For example, if you wanted to know what other options there are for the **`ls`** command, you could type: **`man ls`**.

`cd directory`
Change to **`directory`**. If run without an argument, it changes to the user's home directory. (I.e., equivalent to **`cd ~`**)

`cp name1 name2 or cp namelist directory`
Copy a file or directory. The first version copies one item and possibly renames the copy. In the second version, all items in **`namelist`** are copied into **`directory`** with the same names. A common idiom is to copy all items into the current directory which is indicated by using '**`.`**' (dot). For example,

```
cp /home/droberts/cs215/examples/*.* .
```

will copy all the files in **`/home/droberts/cs215/examples`** to the current directory.

`mv name1 name2 or mv namelist directory`
Move a file or directory. The first version moves one item and can be used to rename it. In the second version, **`namelist`** is a list of items to be moved to **`directory`**.

`more filename`
Display file to screen a page at a time. Typing space will advance to the next page, typing **`b`**' will go back to previous page, typing '**`q`**' quits the program.

`emacs filename or vim filename`
Text editors. The majority of programmers use one of these text editors for creating program source files. Both have vociferous adherents who intensely dislike the other editor. While you can use a another editor (e.g., gedit) if you wish, only emacs and vim are present in nearly every Unix distribution. The two things that everyone needs to know about each is how to save files (**`Ctrl-x Ctrl-s`** in emacs, **`:w<Enter>`** in vim) and how to quit the editor (**`Ctrl-x Ctrl-c`** in emacs, **`:q<Enter>`** in vim).

Both editors have built-in tutorials (**`Ctrl-h t`** from within emacs, **`vimtutor`** on the command line for vim). Links to several on-line tutorials and reference cards are available for each on the course webpage.

`g++ options filenames`
GNU C++ compiler. See separate submission handout regarding use of the compiler.

`make options target`
Generate target from a makefile. See separate handout regarding use of make.

`touch file1 file2 ...`
Changes the last modified date of files to current date. Useful for forcing make to recreate a target by touching a source file.

`rm file1 file2 ...`
Delete files. Can also be used with **`-rf`** options to delete non-empty directories. (The **`rmdir`** command will delete empty directories only.)

`tar options filenames`
An archive facility that combines multiple files into one file. See separate submission handout regarding use of tar.

`jobs`
List programs running in background mode. The job number listed can be used to kill a process using **`%#`**. E.g., **`kill %1`** terminates the process of job 1.

`ps axu`
List all processes running on the machine. When a program runs it has a process associated with it. Occasionally, you'll need to find out the ID of a process (e.g. to terminate a runaway program).

`kill job`
Terminate a process. The job can be specified by PID (process id) or job number (**`%#`**) from the jobs listing. Occasionally, this will not work and you will need to use option **`-9`**, which says to terminate the process if at all possible.

Using Linux in the CS Lab and KC-267
The client machines run X Windows in Linux. Terminal windows can be launched in a variety of ways depending on the GUI, but it can always be done by going to the Applications entry of the main menu, then search for "Terminal". If you are using vim as your editor, you will probably want at least two windows, one for editing and one for compiling. If you are using emacs, you will find that emacs launches its own window. This means that it should be run in the background using:

```
emacs filename &
```

The **`&`** tells the shell to run emacs in the background and returns control to the shell. By doing this, you can now edit in the emacs window and compile in the terminal window.

Using VirtualBox or Wubi
A full Linux distribution may be run in conjunction with Windows on the same machine in two ways. One is to dual-boot as the client machines do. The easiest way to do this is to use Wubi, the Windows Ubuntu Installer, available at http://wubi-installer.org/. The advantage of this method is that Linux is running directly on the hardware, so is a little faster. The disadvantage is that only one operating system is running at time, so the machine must be rebooted to get back to Windows.

Even easier is to run VirtualBox, an x86 virtual machine, available at http://www.virtualbox.org/. The instructor has DVDs available that contain the Windows install program for VirtualBox, a virtual hard disk image (VDI) of Ubuntu 10.04LTS, and instructions on how to set this up. The advantage of this method is that both Windows and Linux are running at the same time. The disadvantage is that the Linux operating system calls are translated into Windows calls, so are a little slower.

Using csserver remotely
To access csserver remotely, a terminal client that supports the SSH (secure shell) protocol must be used so that transmitted data is encrypted. (In particular, telnet and ftp transmit passwords in clear text, so are not supported.) Links to the URLs below also are available on the CS 215 course webpage.

For Windows boxes, Putty is a very small (fits on a USB stick) terminal client (available at http://www.chiark.greenend.org.uk/~sgtatham/putty/). The installation puts an icon on the desktop that can be used to launch Putty. Type csserver.evansville.edu as the machine to connect to, and confirm the protocol being used is SSH before connecting. A dialog box asking about unknown host key will appear (usually only the first time); just click 'Yes'. Then a terminal window will appear and you will be prompted for your username and password. It probably will be convenient to log into the server twice so that you can edit in one window and compile in another.

To transfer files between a local Windows host and csserver (e.g. in order to locally print a file or to submit a project), use PSCP which is available from the Putty website.

For Linux boxes, ssh usually is available. To use it, type the following into a terminal window:

```
ssh username@csserver.evansville.edu
```

where username is your csserver user name. (If your local user name is the same as csserver's, you can leave off the username@ part.) The SSH server on csserver is configured to forward X packets, so you can use your local machine like a CS Lab machine by providing the **`-X`** option to ssh. However, sometimes the UE network is too slow to support interactive graphics and doing plain-text sometimes works better. In particular, if you use emacs, you might want to start it without a new window, which can be accomplished using the **`-nw`** option.

On a local Linux, transfer files to/from csserver using scp which has syntax like cp. Remote names are prefixed with **`username@remotemachinename:`** and are relative to the user home directory. For example,

```
scp username@csserver.evansville.edu:/home/droberts/cs215/examples/*.* .
```

will copy all the files in **`/home/droberts/cs215/examples`** on csserver to the current directory on the local machine. Note that the final period (.) character is part of the command.