1 Introduction

2 Computing Resources
   2.1 The amdpool Servers .................................................. 3
   2.2 ECE Linux Computing Lab (314 Phillips Hall) ...................... 4
   2.3 CIT Windows Computing Lab (318 Phillips Hall) .................. 4
   2.4 Remote Login from Personal Windows Laptop/Workstation .......... 4
   2.5 Remote Login from Personal Mac Laptop/Workstation ............... 4
   2.6 Remote Login from Personal Linux Laptop/Workstation .......... 5
   2.7 Testing X11 ................................................................. 5
   2.8 ECE SSH Rate Limiting .................................................. 6
   2.9 Personal Computing Resources ......................................... 6

3 The Linux Command Line .................................................... 7
   3.1 Changing Your Password ............................................... 7
   3.2 Hello World ................................................................. 7
   3.3 Manual Pages .................................................................. 7
   3.4 Create, View, and List Files ............................................ 7
   3.5 Create, Change, and List Directories .................................. 8
   3.6 Copy, Move, and Remove Files and Directories ...................... 11
   3.7 Using wget to Download Files ......................................... 13
   3.8 Using grep to Search Files ............................................. 13
   3.9 Using find to Find Files ................................................ 14
   3.10 Using tar to Archive Files .............................................. 15
   3.11 Using top to View Running Processes ................................. 16
   3.12 Environment Variables ............................................... 16
   3.13 Command Output Redirection ....................................... 17
   3.14 Command Chaining ..................................................... 17
   3.15 Command Pipelining .................................................... 18
   3.16 Aliases, Wildcards, Command History, and Tab Completion .... 18
4 Linux Text Editors
  4.1 Nano ......................................................... 19
  4.2 Gedit ....................................................... 20
  4.3 Emacs and Vim ............................................. 20
  4.4 MobaTextEditor ............................................ 20

5 The Two-Window Linux Workflow 21

6 ECE 4750 Specific Linux Commands 23
  6.1 ECE 4750 Setup Script .................................. 23
  6.2 Using quota to Check Your Space Usage ................ 23
  6.3 Using trash to Safely Remove Files ..................... 24
  6.4 Restoring Files from a Snapshot ....................... 24

7 Conclusion 24
1. Introduction

All the lab assignments for this course are designed assuming you will be using a Linux (or UNIX-like) operating system for development. Basic Linux knowledge is essential to successfully complete these lab assignments and a more in-depth understanding enhances productivity. This tutorial covers the computing resources to be used in the course and offers a brisk introduction to the Linux operating system for first time users including some details specific to this course.

To follow along with the tutorial, type the commands without the % character. In addition to working through the commands in the tutorial, you should also try the more open-ended tasks marked with the ★ symbol.

2. Computing Resources

If you do not already have an ECE account, then you should register for an account using the following account management webpage:

- https://charon.ece.cornell.edu

Follow the instructions to create an ECE account; it can take a full business day for account creation so please do this as soon as possible. All laboratory assignments will need to be completed using your ECE account.

Note that we will use the Bash shell in this course, and by default all new ECE accounts are setup correctly. If you have an ECE account from a previous course, your shell may be set to something other than Bash. Verify your account is setup correctly by visiting the account management webpage (i.e., the above link). Verify that the Login Shell field says /bin/bash. If it is not, click Change Shell and then Process Changes. You should receive an email, and after a few minutes, you should see the Login Shell is now set to /bin/bash.

There are three ways you can get access to computing resources for the purposes of this course. You can access the ECE computing resources by using the ECE Linux Computing Lab in 314 Phillips Hall, you can use the CIT Windows Computing Lab in 318 Phillips Hall, or you can log into the amdpool servers remotely from your own personal workstation. All the assignments assume that you are working on the ECE computing resources. Since we use open-source tools in this course, it should also be possible for you to directly work locally on your own workstation without logging into the amdpool servers. This would require you to setup your own local development environment, so we cannot really provide too much support for this option.

2.1. The amdpool Servers

The department has a cluster of Linux servers dedicated for ECE instructional use. There are four servers named as follows:

- amdpool-01.ece.cornell.edu
- amdpool-02.ece.cornell.edu
- amdpool-03.ece.cornell.edu
- amdpool-04.ece.cornell.edu

Instead of accessing these servers individually, you will usually just log into amdpool.ece.cornell.edu. This is a special host which helps balance load across the cluster by automatically redirecting you to one of the four servers. These servers run the Red Hat Enterprise Linux 5.11 operating system. We use the term amdpool as short-hand to refer to this cluster of Linux servers.
2.2. ECE Linux Computing Lab (314 Phillips Hall)

You can use the 20+ Linux workstations in 314 Phillips Hall to work on your lab assignments. You should be able to access the lab using your keycard, and you can log into the workstations using your NetID and your ECE account password. Once you have logged into the workstation, you can open a terminal by choosing Applications > Accessories > Terminal from the menu at the top of the desktop.

Note that when you are working in the ECE Linux Computing Lab in 314 Phillips Hall you are actually not logging into amdpool. You are instead working directly on the workstations. The development environment should hopefully be very similar as if you were logging into amdpool remotely.

2.3. CIT Windows Computing Lab (318 Phillips Hall)

Students can also use the 45 Windows workstations in 318 Phillips Hall to log into amdpool remotely. You should be able to access the lab using your keycard, and you do not need a username or password to log into the workstations. You can use MobaXterm to log into amdpool. To start MobaXterm choose All Programs > Class Files > MobaXterm Personal Edition > MobaXterm Personal Edition from the start menu. Once MobaXterm has started you will see it has started a local terminal. Use the following command at the local terminal prompt to log into amdpool.

% ssh -X <netid>@amdpool.ece.cornell.edu

Replace <netid> with your Cornell NetID in the command above, and do not type the % character. We use the % character to indicate what commands we should enter on the command line. Executing the command will prompt you to enter your ECE account password, and then you should be connected to amdpool. The -X command line option enables X11 forwarding so that we can start a GUI application like gedit or gtkwave on the server yet have the actual GUI displayed on our local machine.

After logging into amdpool you will notice that a sidebar appears on the left that shows you the files in your home directory. You can drag files to/from the sidebar and the desktop to easily move files to/from the Windows workstation and amdpool. To hide the sidebar choose View > Show/hide sidebar from the menu.

2.4. Remote Login from Personal Windows Laptop/Workstation

You can also log into amdpool from your own personal Windows workstation. You will need to install MobaXterm on your workstation. Download the installer from here:

• http://mobaxterm.mobatek.net/MobaXterm_Setup_8.1.msi

Run the installer, click next to start the installation, agree to the EULA and click next, click next to install in the default location, click install to start the installation, and click finish to finish the installation. Then start MobaXterm by choosing All Programs > MobaXterm Personal Edition > MobaXterm Personal Edition from the start menu. You should agree if Windows asks to grant any additional security permissions to MobaXterm. Then follow the directions in Section 2.3. Note that if your connect to amdpool seems to be dropping, you may need to set the SSH “keep alive signal”. Choose Settings > Configuration and check the option to send a keep alive signal every 60 seconds.

2.5. Remote Login from Personal Mac Laptop/Workstation

To start, you need to open a local terminal by typing “terminal” into Spotlight. By default your connection to amdpool will be dropped if you are inactive for a certain amount of time. To prevent
this, you need to add some configuration information to a specific file. We will learn more about the 
\texttt{echo} command and command output redirection later in the tutorial, but for now carefully enter the 
following commands in the terminal.
\begin{verbatim}
% echo "Host *.ece.cornell.edu" >> ~/.ssh/config
% echo " ServerAliveInterval 180" >> ~/.ssh/config
\end{verbatim}

Do not type the \% character. We use the \% character to indicate what commands we should enter on 
the command line. You will need to install X11, although some older versions of Mac OS X have X11 
installed by default, or you may have previously installed X11. To see if you have X11 installed, enter 
the following command in the terminal.
\begin{verbatim}
% xclock
\end{verbatim}

Again, do not type the \% character. If your system cannot find \texttt{xclock} then you need to install X11. 
The specific X11 window system you will want to use is called XQuartz, and it is available from here:
\begin{itemize}
\item \url{http://xquartz.macosforge.org/landing}
\end{itemize}

Use the DMG to install XQuartz and then try running \texttt{xclock} again. Once \texttt{xclock} works locally, then 
use SSH to log into \texttt{amdpool} as follows:
\begin{verbatim}
% ssh -X <netid>@amdpool.ece.cornell.edu
\end{verbatim}

Replace \texttt{<netid>} with your Cornell NetID in the command above and do not type the \% character.

2.6. Remote Login from Personal Linux Laptop/Workstation

If you are using Linux you will likely know how to open a local terminal. By default your connec-
tion to \texttt{amdpool} will be dropped if you are inactive for a certain amount of time. To prevent this, 
you need to add some configuration information to a specific file. We will learn more about the 
\texttt{echo} command and command output redirection later in the tutorial, but for now carefully enter the 
following commands in the terminal.
\begin{verbatim}
% echo "Host *.ece.cornell.edu" >> ~/.ssh/config
% echo " ServerAliveInterval 180" >> ~/.ssh/config
\end{verbatim}

Then use SSH as follows:
\begin{verbatim}
% ssh -X <netid>@amdpool.ece.cornell.edu
\end{verbatim}

As mentioned above, replace \texttt{<netid>} with your Cornell NetID in the command above and do not 
type the \% character.

2.7. Testing X11

After logging into \texttt{amdpool} remotely using any of the above approaches, you can verify that a GUI 
application (e.g., \texttt{gedit} or \texttt{gtkwave}) will work by running the following command.
\begin{verbatim}
% xclock
\end{verbatim}
Just to be very explicit, you want to execute the `xclock` command remotely after logging into `amdpool` via SSH (i.e., not locally on your Mac/Linux laptop/workstation). If you cannot see the analog clock, there is likely an issue with your X11 setup. You can also try the more interesting `xeyes`.

```bash
% xeyes
```

### 2.8. ECE SSH Rate Limiting

The ECE network includes SSH rate-limiting at the firewall. If you try and SSH into an ECE server from outside the ECE network more than ten times in a minute it will block you for some period of time (potentially up to an hour). So if you suddenly cannot log into `amdpool`, you might want to try changing workstations or waiting a while before trying again.

### 2.9. Personal Computing Resources

While we strongly encourage students to use the ECE computing resources, more advanced students are welcome to work directly on their own workstations without logging into `amdpool`. This is not too much work if you are using a UNIX-like system (e.g., Mac OS X, Linux). It will require the student to install `iverilog`, `gtkwave`, and the PyMTL hardware modeling framework. More information about installing these tools is available at the following links:

- [http://iverilog.icarus.com](http://iverilog.icarus.com)
- [http://gtkwave.sourceforge.net](http://gtkwave.sourceforge.net)
- [https://github.com/cornell-brg/pymtl](https://github.com/cornell-brg/pymtl)

The course staff cannot offer too much support for this, and please remember that all lab submissions must work on `amdpool` since that is where the course staff will be doing the assessment.

### 3. The Linux Command Line

In this section, we introduce the basics of working at the Linux command line. Please note that this Linux tutorial is obviously not comprehensive and cannot replace the extensive amount of documentation available online or elsewhere. The goal is to get you comfortable with commands required to complete the lab assignments. Before trying the commands listed in this section, you will need to get access to the ECE computing resources as described in the previous section.

The shell is the original Linux user interface which is a text-based command-line interpreter. The default shell once you log into `amdpool` is Bash. While there are other shells such as `sh`, `csh`, and `tcsh`, for this course we will always be assuming you are using Bash. As mentioned above, we use the `%` character to indicate commands that should be entered at the Linux command line, but you should not include the actual `%` character when typing in the commands on your own. To make it easier to cut-and-paste commands from this tutorial document onto the command line, you can tell Bash to ignore the '%` character using the following command:

```bash
% alias %=""
```

Now you can cut-and-paste a sequence of commands from this tutorial document and Bash will not get confused by the `%` character which begins each line.
3.1. Changing Your Password

You should also change your password after logging into amdpool. You can do this by entering the `passwd` command at the shell prompt like this:

```
% passwd
```

You should choose a strong password which uses a long combination of uppercase letters, lowercase letters, numbers, and symbols. The system places constraints on what you choose to ensure that you pick a strong password.

3.2. Hello World

We begin with the ubiquitous “Hello, World” example. To display the message “Hello, World” we will use the `echo` command. The `echo` command simply “echos” its input to the console.

```
% echo "Hello, World"
```

The string we provide to the `echo` command is called a command line argument. We use command line arguments to tell commands what they should operate on. Although simple, the `echo` command can very useful for creating simple text files, displaying environment variables, and general debugging.

★ To-Do On Your Own: Experiment with using the `echo` command to display different messages.

3.3. Manual Pages

You can learn more about any Linux command by using the `man` command. Try using this to learn more about the `echo` command.

```
% man echo
```

You can use the up/down keys to scroll the manual one line at a time, the space bar to scroll down one page at a time, and the q key to quit viewing the manual. You can even learn about the `man` command itself by using `man man`. As you follow the tutorial, feel free to use the `man` command to learn more about the commands we cover.

★ To-Do On Your Own: Use the `man` command to learn more about the `passwd` command.

3.4. Create, View, and List Files

We can use the `echo` command and a feature called command output redirection to create simple text files. We will discuss command output redirection in more detail later in the tutorial. Command output redirection uses the > operator to take the output from one command and “redirect” it to a file. The following commands will create a new file named `ece4750-tut1.txt` that simply contains the text “Computer Architecture”.

```
% echo "Computer Architecture" > ece4750-tut1.txt
```

We can use the `cat` command to quickly display the contents of a file.
% cat ece4750-tut1.txt

For larger files, cat will output the entire file to the console so it may be hard to read the file as it streams past. We can use the less command to show one screen-full of text at a time. You can use the up/down keys to scroll the file one line at a time, the space bar to scroll down one page at a time, and the q key to quit viewing the file.

% less ece4750-tut1.txt

You can use the ls command to list the filenames of the files you have created.

% ls

We can provide command line options to the ls command to modify the command’s behavior. For example, we can use the -l (i.e., a dash followed by the number one) command line option to list one file per line, and we can use the -l (i.e., a dash followed by the letter l) command line option to provide a longer listing with more information about each file.

% ls -l

You should see the newly created ece4750-tut1.txt file along with some additional directories or folders. We will discuss directories in the next section. Use the following commands to create a few more files using the echo command and command output redirection, and then list the files again.

% echo "Application" > ece4750-tut1-layer1.txt
% echo "Algorithm" > ece4750-tut1-layer2.txt
% ls -l

★ To-Do On Your Own: Create a new file named ece4750-tut1-layer3.txt which contains the third layer in the computing systems stack (i.e., programming language). Use cat and less to verify the file contents.

3.5. Create, Change, and List Directories

Obviously, having all files in a single location would be hard to manage effectively. We can use directories (also called folders) to logically organize our files, just like one can use physical folders to organize physical pieces of paper. The mechanism for organizing files and directories is called the file system. When you first log into amdpool, you will be in your home directory. This is your own private space on the server that you can use to work on the lab assignments and store your files. You can use the pwd command to print the directory in which you are currently working, which is known as the current working directory.

% pwd
/home/students/<netid>

You should see output similar to what is shown above, but instead of students it may say graduate if you are a graduate student and instead of <netid> it should show your actual NetID. The pwd command shows a directory path. A directory path is a list of nested directory names; it describes a “path” to get to a specific file or directory. So the above path indicates that there is a toplevel
directory named home, that contains a directory named students, that contains a directory named <netid>. This is the directory path to your home directory. As an aside, notice that Linux uses a forward slash (/) to separate directories, while Windows uses a back slash (\) for the same purpose.

We can use the mkdir command to make new directories. The following command will make a new directory named ece4750 within your home directory.

```
% mkdir ece4750
```

We can use the cd command to change our current working directory. The following command will change the current working directory to be the newly created ece4750 directory, before displaying the current working directory with the pwd command.

```
% cd ece4750
% pwd
/home/students/<netid>/ece4750
```

Use the mkdir, cd, and pwd commands to make another directory.

```
% mkdir tut1
% cd tut1
% pwd
/home/students/<netid>/ece4750/tut1
```

We sometimes say that tut1 is a subdirectory or a child directory of the ece4750 directory. We might also say that the ece4750 directory is the parent directory of the tut1 directory.

There are some important shortcuts that we can use with the cd command to simplify navigating the file system. The special directory named . (i.e., one dot) always refers to the current working directory. The special directory named .. (i.e., two dots) always refers to the parent of the current working directory. The special directory named ~ (i.e., a tilde character) always refers to your home directory. The special directory named / (e.g., single forward slash) always refers to the highest-level root directory. The following commands illustrate how to navigate up and down the directory hierarchy we have just created.

```
% pwd
/home/students/<netid>/ece4750/tut1
% cd .
% pwd
/home/students/<netid>/ece4750/tut1
% cd..
% pwd
/home/students/<netid>/ece4750
% cd..
% pwd
/home/students/<netid>
% cd ece4750/tut1
% pwd
/home/students/<netid>/ece4750/tut1
% cd
% pwd
/home/students/<netid>
```
Notice how we can use the cd command to change the working directory to another arbitrary directory by simply using a directory path (e.g., ece4750/tut1). These are called relative paths because the path is relative to your current working directory. You can also use an absolute path which always starts with the root directory to concretely specify a directory irrespective of the current working directory. A relative path is analogous to directions to reach a destination from your current location (e.g., How do I get to the coffee shop from my current location?), while an absolute path is analogous to directions to reach a destination from a centralized location (e.g., How do I get to the coffee shop from the center of town?).

This example illustrates one more useful shortcut. The cd command with no command line arguments always changes the current working directory to your home directory. We can use the ls command to list files as well as directories. Use the following commands to create a new file and directory in the ece4750/tut1 subdirectory, and then list the file and directory.

You should see both the dirA subdirectory and the newly created ece4750-tut1.txt file listed. Feel free to use the cat command to verify the file contents of the newly created file. We can use the tree command to recursively list the contents of a directory. The following commands create a few more directories before displaying the directory hierarchy.
% tree
.
+-- dirA
  +-- dirB
  |   +-- dirB_1
  |   +-- dirB_2
  |-- dirC
  |   +-- dirC_1
  '-- ece4750-tut1.txt

Note that we are using the `-p` command line option with the `mkdir` command to make multiple nested directories in a single step.

★ To-Do On Your Own: Experiment with creating additional directories and files within the `ece4750/tut1` subdirectory. Try creating deeper hierarchies with three or even four levels of nesting using the `-p` option to the `mkdir` command. Experiment with using the `.` and `..` special directories. Use the `tree` command to display your newly created directory hierarchy.

### 3.6. Copy, Move, and Remove Files and Directories

We can use the `cp` command to copy files. The first argument is the name of the file you want to copy, and the second argument is the new name to give to the copy. The following commands will make two copies of the files we created in the previous section.

```
% cd ~/ece4750/tut1
% cp ece4750-tut1.txt ece4750-tut1-a.txt
cpy ece4750-tut1.txt ece4750-tut1-b.txt
% ls -l
```

We can also copy one or more files into a subdirectory by using multiple source files and a final destination directory as the arguments to the `cp` command.

```
% cd ~/ece4750/tut1
% cp ece4750-tut1.txt dirA
% cp ece4750-tut1-a.txt ece4750-tut1-b.txt dirA
% tree
```

We can use the `-r` command line option to enable the `cp` command to recursively copy an entire directory.

```
% cd ~/ece4750/tut1
% tree
% cp -r dirA dirD
% tree
```

If we want to move a file or directory, we can use the `mv` command. As with the `cp` command, the first argument is the name of the file you want to move and the second argument is the new name of the file.
Again, similar to the `cp` command, we can also move one or more files into a subdirectory by using multiple source files and a final destination directory as the arguments to the `mv` command.

```
% cd ~/ece4750/tut1
% tree
% mv ece4750-tut1-a.txt dirB
% mv ece4750-tut1-b.txt ece4750-tut1-c.txt dirB
% tree
```

We do not need to use the `-r` command line option to move an entire directory at once.

```
% cd ~/ece4750/tut1
% tree
% mv dirD dirE
% tree
```

The following example illustrates how we can use the special `.` directory to move files from a subdirectory into the current working directory.

```
% cd ~/ece4750/tut1
% tree
% mv dirE/ece4750-tut1.txt .
% tree
```

We can use the `rm` command to remove files. The following command removes a file from within the `ece4750/tut1` subdirectory.

```
% cd ~/ece4750/tut1
% ls -1
% rm ece4750-tut1.txt
% ls -1
```

To clean up, we might want to remove the files we created in your home directory earlier in this tutorial.

```
% cd
% rm ece4750-tut1.txt
% rm ece4750-tut1-layer1.txt
% rm ece4750-tut1-layer2.txt
% rm ece4750-tut1-layer3.txt
```

We can use the `-r` command line option with the `rm` command to remove entire directories, but please be careful because it is relatively easy to permanently delete many files at once. See Section 6.3 for a useful command that you might want to use instead of the `rm` command to avoid accidentally deleting important work. Also see Section 6.4 for information about a nice snapshotting feature available on `amdpool`.
% cd ~/ece4750/tut1
% ls -1
% rm -r dirA dirB dirC dirE
% ls -1

★ To-Do On Your Own: Creating additional directories and files within the ece4750/tut1 subdirectory, and then use the cp, mv, and rm commands to copy, move, and remove the newly created directories and files. Use the ls and tree commands to display your file and directory organization.

3.7. Using wget to Download Files

We can use the wget command to download files from the internet. For now, this is a useful way to retrieve a text file that we can use in the following examples.

% cd ~/ece4750/tut1
% wget http://www.csl.cornell.edu/courses/ece4750/overview.txt
% cat overview.txt

3.8. Using grep to Search Files

We can use the grep command to search and display lines of a file that contain a particular pattern. The grep command can be useful for quickly searching the contents of the source files in your lab assignment. The command takes the pattern and the files to search as command line arguments. The following command searches for the word “memories” in the overview.txt file downloaded in the previous section.

% cd ~/ece4750/tut1
% grep "memories" overview.txt

You should see just the three lines within the overview.txt file that contain the word “memories”. We can use the --line-number and --color command line options with the grep command to display the line number of each match and to highlight the matched word.

% cd ~/ece4750/tut1
% grep --line-number --color "memories" overview.txt

We can use the -r command line option to recursively search all files within a given directory hierarchy. In the following example, we create a subdirectory, copy the overview.txt file, and illustrate how we can use the grep command to recursively search for the word “memories”.

% cd ~/ece4750/tut1
% mkdir dirA
% cp overview.txt dirA
% grep -r --line-number --color "memories" .

Notice how we specify a directory as a command line argument (in this case the special . directory) to search the current working directory. You should see the three lines from both copies of the overview.txt file. The grep command also shows which file contains the match.
As another example, we will search two special files named `/proc/cpuinfo` and `/proc/meminfo`. These files are present on every modern Linux system, and they contain information about the processor and memory hardware in that system. The following command first uses the `less` command so you can browse the file, and then uses the `grep` command to search for processor in the `/proc/cpuinfo` file. Recall that with the `less` command, we use the up/down keys to scroll the file one line at a time, the space bar to scroll down one page at a time, and the q key to quit viewing the file.

```bash
% cd ~/ece4750/tut1
% less /proc/cpuinfo
% grep "processor" /proc/cpuinfo
```

It should be pretty clear that you are using a multicore processor! You can also search to find out which company makes the processors and what clock frequency they are running at:

```bash
% cd ~/ece4750/tut1
% grep "vendor_id" /proc/cpuinfo
% grep "cpu MHz" /proc/cpuinfo
```

We can find out how much DRAM is in the system by searching for `MemTotal` in the `/proc/meminfo` file.

```bash
% cd ~/ece4750/tut1
% grep "MemTotal" /proc/meminfo
```

★ **To-Do On Your Own:** Try using `grep` to search for the words “processor” and “network” in the `overview.txt` file.

### 3.9. Using `find` to Find Files

We can use the `find` command to recursively search a directory hierarchy for files or directories that match a specified criteria. While the `grep` command is useful for searching file contents, the `find` command is useful for quickly searching the file and directory names in your lab assignments. The `find` command is very powerful, so we will just show a very simple example. First, we create a few new files and directories.

```bash
% cd ~/ece4750/tut1
% mkdir -p dirB/dirB_1
% mkdir -p dirB/dirB_2
% mkdir -p dirC/dirC_1
% echo "test" > dirA/file0.txt
% echo "test" > dirA/file1.txt
% echo "test" > dirB/dirB_1/file0.txt
% echo "test" > dirB/dirB_1/file1.txt
% echo "test" > dirB/dirB_2/file0.txt
% echo "test" > dirB/dirB_2/file1.txt
% tree
```

We will now use the `find` command to find all files named "file0.txt". The `find` command takes one command line argument to specify where we should search and a series of command line options.
to describe what files and directories we are trying to find. We can also use command line options
to describe what action we would like to take when we find the desired files and directories. In
this example, we use the `--name` command line option to specify that we are searching for files with
a specific name. We can also use more complicated patterns to search for all files with a specific
filename prefix or extension.

```bash
% cd ~/ece4750/tut1
% find . -name "file0.txt"
```

Notice that we are using the special `. `directory to tell the `find` command to search the current
working directory and all subdirectories. The `find` command always searches recursively.

**To-Do On Your Own:** Create additional files named "file2.txt" in some of the subdirectories we
have already created. Use the "find" command to search for files named "file2.txt".

### 3.10. Using tar to Archive Files

We can use the `tar` command to “pack” files and directories into a simple compressed archive, and
also to “unpack” these files and directories from the archive. This kind of archive is sometimes
called a tarball. Most open-source software is distributed in this compressed form. It makes it easy
to distribute code among collaborators and it is also useful to create backups of files. We can use
the following command to create an archive of our tutorial directory and then remove the tutorial
directory.

```bash
% cd ~/ece4750
% tar -czvf tut1.tgz tut1
% rm -r tut1
% ls -l
```

Several command line options listed together as a single option (-czvf), where c specifies we want
to create an archive, z specifies we should use “gzip” compression, v specifies verbose mode, and f
specifies we will provide filenames to archive. The first command line argument is the name of the
archive to create, and the second command line argument is the directory to archive. We can now
extract the contents of the archive to recreate the tutorial directory. We also remove the archive.

```bash
% cd ~/ece4750
% tar -xzvf tut1.tgz
% rm tut1.tgz
% tree tut1
```

Note that we use the x command line option with the `tar` command to specify that we intend to
extract the archive.

**To-Do On Your Own:** Create an example directory within the ece4750/tut1 subdirectory. Copy
the `overview.txt` file and rename it to add example files to your new directory. Use the `tar`
command to create and extract an archive of just this one new directory.
3.11. Using *top* to View Running Processes

You can use the *top* command to view what commands are currently running on the Linux system in realtime. This can be useful to see if there are many commands running which are causing the system to be sluggish. When finished you can use the q character to quit.

```
% top
```

The first line of the *top* display shows the number of users currently logged into the system, and the *load average*. The load average indicates how “overloaded” the system was over the last one, five, and 15 minutes. If the load average is greater than the number of processors in the system, it means your system will probably be sluggish. You can always try logging into a different server in the cluster.

3.12. Environment Variables

In the previous sections, we have been using the Bash shell to run various commands, but the Bash shell is actually a full-featured programming language. One aspect of the shell that is similar in spirit to popular programming languages, is the ability to write and read *environment variables*. The following commands illustrate how to write an environment variable named *ece4750_tut1_layer1*, and how to read this environment variable using the *echo* command.

```
% ece4750_tut1_layer1="application"
% echo ${ece4750_tut1_layer1}
```

Keep in mind that the names of environment variables can only contain letters, numbers, and underscores. Notice how we use the ${} syntax to read an environment variable. There are a few built-in environment variables that might be useful:

```
% echo ${HOSTNAME}
% echo ${HOME}
% echo ${PWD}
```

We often use the HOME environment variable in directory paths like this:

```
% cd ${HOME}/ece4750
```

The PWD environment variable always holds the current working directory. We can use environment variables as options to commands other than *echo*. A common example is to use an environment variable to “remember” a specific directory location, which we can quickly return to with the cd command like this:

```
% cd ${HOME}/ece4750/tut1
% TUT1=${PWD}
% cd
% pwd
/home/students/<netid>
% cd ${TUT1}
% pwd
/home/students/<netid>/ece4750/tut1
```
**To-Do On Your Own:** Create a new environment variable named ece4750_tut1_layer2 and write it with the second layer in the computer systems stack (i.e., algorithm). Use the `echo` command to display this environment variable. Experiment with creating a new subdirectory within ece4750/tut1 and then using an environment variable to “remember” that location.

### 3.13. Command Output Redirection

We have already seen using the `echo` command and command output redirection to create simple text files. Here is another example:

```
% cd ${HOME}/ece4750/tut1
% echo "Application" > computing-stack.txt
% cat computing-stack.txt
```

The `>` operator tells the Bash shell to take the output from the command on the left and overwrite the file named on the right. We can use any command on the left. For example, we can save the output from the `pwd` command or the `man` command to a file for future reference.

```
% cd ${HOME}/ece4750/tut1
% pwd > cmd-output.txt
% cat cmd-output.txt
% man pwd > cmd-output.txt
% cat cmd-output.txt
```

We can also use the `>>` operator which tells the Bash shell to take the output from the command on the left and append the file named on the right. We can use this to create multiline text files:

```
% cd ${HOME}/ece4750/tut1
% echo "Algorithm" > computing-stack.txt
% echo "Programming Language" >> computing-stack.txt
% echo "Operating System" >> computing-stack.txt
% cat computing-stack.txt
```

**To-Do On Your Own:** Add the remaining levels of the computing stack (i.e., gate-level, circuits, devices, technology) to the `computing-stack.txt` text file. Use the `cat` command to verify that the file contents.

### 3.14. Command Chaining

We can use the `&&` operator to specify two commands that we want to chaining together. The second command will only execute if the first command succeeds. Below is an example.

```
% cd ${HOME}/ece4750/tut1 && cat computing-stack.txt
```

**To-Do On Your Own:** Create a single-line command that combines creating a new directory with the `mkdir` command and then immediately changes into the directory using the `cd` command.
3.15. Command Pipelining

The Bash shell allows you to run multiple commands simultaneously, with the output of one command becoming the input to the next command. We can use this to assemble “pipelines”; we “pipe” the output of one command to another command for further actions using the `|` operator.

The following example uses the `grep` command to search the special `proc/cpuinfo` file for lines containing the word “processor” and then pipes the result to the `wc` command. The `wc` command counts the number of characters, words, or lines of its input. We use the `-l` command line option with the `wc` command to count the number of lines.

```
% grep processor /proc/cpuinfo | wc -l
```

This is a great example of the Linux philosophy of providing many simple commands that can be combined to create more powerful functionality. Essentially the pipeline we have created is a command that tells us the number of processors in our system.

As another example, we will pipe the output of the `last` command to the `grep` command. The `last` command lists the names of all of the users that have logged into the system since the system was rebooted. We can use `grep` to search for your NetID and thus quickly see how when you previously have logged into this system.

```
% last | grep <netid>
```

We can create even longer pipelines. The following pipeline will report the number of times you have logged into the system since it was rebooted.

```
% last | grep <netid> | wc -l
```

**To-Do On Your Own:** Use the `cat` command with the `overview.txt` file and pipe the output to the `grep` command to search for the word “memories”. While this is not as fast as using `grep` directly on the file, it does illustrate how many commands (e.g., `grep`) can take their input specified as a command line argument or through a pipe.

3.16. Aliases, Wildcards, Command History, and Tab Completion

In this section, we describe some miscellaneous features of the Bash shell which can potentially be quite useful in increasing your productivity.

Aliases are a way to create short names for command sequences to make it easier to quickly execute those command sequences in the future. For example, assume that you frequently want to change to a specific directory. We can create an alias to make this process take just two keystrokes.

```
% alias ct="cd ${HOME}/ece4750/tut1"
% ct
% pwd
/home/academic/<netid>/ece4750/tut1
```

If you always want this alias to be available whenever you log into the system, you can save it in your `.bashrc` file. The `.bashrc` is a special Bash script that is run on every invocation of a Bash shell.
The reason we have to use a back slash (\) in front of the double quotes is to make sure the echo command sees this command line argument as one complete string.

Wildcards make it easy to manipulate many files and directories at once. Whenever we specify a file or directory on the command line, we can often use a wildcard instead. In a wildcard, the asterisk (*) will match any sequence of characters. The following example illustrates how to list all files that end in the suffix .txt and then copies all files that match the wildcard from one directory to another.

```
% cd ${HOME}/ece4750/tut1
% ls *.txt
% cp dirA/file*.txt dirB
% tree
```

The Bash shell keeps a history of everything you do at the command line. You can display the history with the history command. To rerun a previous command, you can use the ! operator and the corresponding command number shown with the history command.

```
% history
% history | grep wc
```

If you press the up arrow key at the command line, the Bash shell will show you the previous command you used. Continuing to press the up/down keys will enable you to step through your history. It is very useful to press the up arrow key once to rerun your last command.

The Bash shell supports tab completion. When you press the tab key twice after entering the beginning of a filename or directory name, Bash will try to automatically complete the filename or directory name. If there is more than one match, Bash will show you all of these matches so you can continue narrowing your search.

### 4. Linux Text Editors

You will need to use a text editor to edit source files in Linux. There are two kinds of text editors: graphical and non-graphical. The non-graphical text editors work by opening files through the command line and then using the keyboard to navigate files, execute commands, etc. The graphical text editors work by providing a GUI so that the user can use a mouse to interact with the editor.

#### 4.1. Nano

Nano is a very simple non-graphical text editor installed on amdpool. The editor is easy to learn and use. You can start Nano by typing the command nano in the terminal and optionally specifying the filename you want to view and edit.

```
% cd ${HOME}
% nano /classes/ece4750/misc/ece4750-overview.txt
```
Use the arrow keys to move the cursor position. Notice that the editor specifies most of the useful commands at the bottom of the terminal screen. The symbol `^` indicates the CONTROL key. To type any text you want, just move the cursor to the required position and use the keyboard. To save your changes press CONTROL+O and press the <ENTER> key after specifying the filename you want to save to. You can exit by pressing CONTROL+X.

---

**To-Do On Your Own:** Use Nano to make some changes to the ece4750-overview.txt text file and then save your edits to your home directory. View the new file using the cat command from the command line and then delete the file using the rm command.

---

### 4.2. Gedit

Gedit is a simple graphical text editor in Linux. You can start Gedit by typing the command `gedit` in the terminal and optionally specifying the filename you want to view and edit.

```
% cd ${HOME}
% gedit /classes/ece4750/misc/ece4750-overview.txt &
```

This should open a new GUI window. If you get an error about your display, you probably do not have your X11 server setup correctly (see Section 2). Navigating, editing, and saving files is very simple in Gedit. Note that we have added a & symbol at the end of the command. This special symbol starts the graphical program in the background enabling you to continue to use the terminal.

See Figure 1 for a screenshot of a typical Gedit session. Notice how the side pane is showing the integrated file browser. You can display the file browser by clicking on the small tab in the bottom left-hand corner of the window. Double-clicking on a file in the file browser will open the file in a new tab. Notice how two tabs are opened: one for the Verilog design file and the other for the Verilog test harness. By default, Gedit will use syntax highlighting for the C programming language. To select Verilog syntax highlighting for the current tab, choose `View > Highlight Mode > Source > Verilog` from the menu.

---

**To-Do On Your Own:** Use Gedit to make some changes to the ece4750-overview.txt text file and then save your edits to your home directory. View the new file using the cat command from the command line and then delete the file using the rm command.

---

### 4.3. Emacs and Vim

While nano and gedit editors are easy to learn, students that anticipate using Linux in the future beyond this course might want to use a more powerful editor such as emacs, vi, or vim. It is beyond the scope of this tutorial to teach you the usage of these editors, but most advanced Linux users use one of these more powerful text editors for development.

### 4.4. MobaTextEditor

If you are using MobaXterm, you can simply right click on a file in the integrated MobaXterm file browser and choose “Open with Default Text Editor” to open the file with the embedded MobaTextEditor. Unlike the other text editors mentioned in this section, the MobaTextEditor actually runs locally on your Windows workstation. MobaXterm ensures that any changes you make to a file
through the MobaTextEditor are automatically uploaded to amdPOOL. You might need to experiment to determine the most productive setup.

5. The Two-Window Linux Workflow

Some students use a one-window workflow. They use various commands at the command line and whenever they want to edit a file they launch a text editor (e.g., gedit), edit and save the file, exit the text editor, and then return to working at the command line. This can be a tedious process and involve many keystrokes and/or mouse clicks to simply edit a file and see the corresponding effect.

We strongly encourage students to use a two-window workflow regardless of how they are accessing the ECE computing resources and which text editor they are using. A two-window workflow involves always having two windows side-by-side. The window on the left will be a terminal at the command line, while the window on the right will be a text editor. The student should be able to switch back-and-forth between the two windows using the keyboard although this is not strictly necessary. By using two windows, the student can work at the command line, quickly switch to edit/save a file, and then quickly switch back to see the corresponding effect.

Figure 2 illustrates an example two-window workflow on the workstations in the ECE Linux Computing Lab in 314 Phillips Hall. A terminal is on the left and gedit is on the right. You can use
Figure 2: Recommended Two-Window Workflow for Linux

Figure 3: Recommended Two-Window Workflow for Windows
the ALT-TAB keyboard combination to quickly switch back-and-forth between the two windows. Figure 3 illustrates an example two-window workflow on the workstations in the CIT Windows Computing Lab in 318 Phillips Hall. MobaXterm is on the left and gedit is on the right. Again, you can use the ALT-TAB keyboard combination to quickly switch back-and-forth between the two windows. Windows has a nice feature where if you drag a window off the left or right side of the screen it will automatically make the window fill just the left or right half of the screen. The key is to be able to be able to see the command line and your text editor at the same time, and to also be able to quickly switch back-and-forward between the command line and your text editor.

6. ECE 4750 Specific Linux Commands

In this section, we describe various aspects of the development environment that are specific to the servers used in the course.

6.1. ECE 4750 Setup Script

Once you are logged into an amdpool machine, as explained in Section 2, you will need to setup the working environment with the following command in order to work on the ECE 4750 lab assignments.

```
% source setup-ece4750.sh
```

The `source` command executes the commands in the given file. Running the command will display some information about what the setup script is doing. Since we always need to source the setup script, we can add this to our `.bashrc` file as follows.

```
% echo "source setup-ece4750.sh -q" >> ${HOME}/.bashrc
```

The extra `-q` command line option prevents the script from displaying its output every time we login to amdpool. With these modifications to our `.bashrc`, we know that the environment will be correctly setup every time we login.

If for any reason running the setup script prevents you from using tools for another course, you will need to run the setup script manually every time you want to work on an ECE 4750 lab assignment.

6.2. Using `quota` to Check Your Space Usage

Students are allocated 1GB of storage on the servers. You can check how much space you are using on the account management webpage:

- `https://charon.ece.cornell.edu`

The `Current Usage` field shows how much space you are using and the `Current Limit` field shows your quota. You can also use the following command to show much space you are using, but also to show the top 20 largest files or directories.

```
% quota
```

Note that it can take 20–30 seconds for this command to finish, so please be patient. If you have exceed the 1GB quota, you can list the size of each file or directory at the top-level of your home directory like this:
% cd ${HOME}
% du -sh *

By recursively changing directories and examining the sizes of files and directories you can figure out what you need to delete.

6.3. Using trash to Safely Remove Files

We have installed a simple program called trash which moves files you wish to delete into a special subdirectory of your home directory located at ${HOME}/tmp/trash. The following commands create a file and then deletes it using trash.

% cd ${HOME}
% echo "This file will be deleted." > testing.txt
% trash testing.txt
% echo "This file will also be deleted." > testing.txt
% trash testing.txt
% ls ${HOME}/tmp/trash

If you look in ${HOME}/tmp/trash you will see subdirectories organized by date. Look in the subdirectory with today’s date and you should two files corresponding to the two files you deleted. We highly recommend always using the trash command instead of rm since this avoids accidentally deleting your work.

6.4. Restoring Files from a Snapshot

The file system used on amdpool has a nice feature where it will automatically create snapshots of your home directory every hour for the last six hours, every night for the last week, and every week for the last two weeks. To access these snapshots simple change directories into ${HOME}/.snapshot and browse around.

% cd ${HOME}/.snapshot
% ls

This can be a life saver if you did not use the trash command and accidentally deleted something important, or if you have edited some files but wish to revert back to an earlier version.

7. Conclusion

This tutorial hopefully helped you become familiar with Linux and how to use it for working on the labs. You have gained some experience working at the command line and also working with either a graphical or non-graphical text editor. We have introduced the two-window Linux workflow and some Linux commands specific to ECE 4750. There are many more resources online for learning Linux, and keep in mind that learning to work productively using the Linux operating system can pay dividends in many other contexts besides this course.

Acknowledgments

This tutorial was developed for the ECE 4750 Computer Architecture course at Cornell University by Shreesha Srinath, Christopher Torng, and Christopher Batten.