ECE 4750 Computer Architecture
Tutorial 1: Linux Development Environment

School of Electrical and Computer Engineering
Cornell University

revision: 2022-08-23-22-00

Contents

1. Introduction

The laboratory 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 this work 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.

Before you begin, make sure that you have **logged into the `ecelinux` servers** as described in the remote access tutorial. You will need to open a terminal and be ready to work at the Linux command line. You can do this using any of the methods described in the remote access tutorial: (1) Windows PowerShell or Mac OS X Terminal; (2) VS Code; or (3) X2Go. 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.

Before you begin, make sure that you have **sourced the setup-ece4750.sh script** or that you have added it to your `.bashrc` script, which will then source the script every time you login. Sourcing the setup script sets up the environment required for this tutorial.

2. The Linux Command Line

In this section, we introduce the basics of working at the Linux command line. Our goal is to get you comfortable with commands required to complete the laboratory assignments. The shell is the original Linux user interface which is a text-based command-line interpreter. The default shell on the `ecelinux` machines is Bash. While there are other shells such as `sh`, `csh`, and `tcsh`, for this course we will always assume 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:

```
% 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.

2.1. 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 "echoes" 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.

2.2. 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 `cat` command.

2.3. 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 `-1` (i.e., a dash followed by the number one) command line option to list one file per line, and we can 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 -1
 % 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 -1
```

★ *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.

2.4. 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 login to an `ecelinux` machine, you will be in your *home directory*. This is your own private space on the server that you can use to work on the laboratory 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/netid
```

You should see output similar to what is shown above, but 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 `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/netid/ece4750
```

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

```
% mkdir tut1
% cd tut1
% pwd
/home/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/netid/ece4750/tut1
% cd .
% pwd
/home/netid/ece4750/tut1
% cd ..
% pwd
/home/netid/ece4750
% cd ..
% pwd
/home/netid
% cd ece4750/tut1
% pwd
/home/netid/ece4750/tut1
% cd
% pwd
/home/netid
% cd /
% pwd
/
% cd ~/ece4750
% pwd
/home/netid/ece4750
```

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?).

```
% pwd
/home/netid/ece4750
% cd /home/netid/ece4750/tut1
% pwd
/home/netid/ece4750/tut1
% cd
% pwd
/home/netid
```

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.

```
% cd ~/ece4750/tut1
% echo "Computer Systems Programming" > ece4750-tut1.txt
% mkdir dirA
% ls -1
```

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.

```
% cd ~/ece4750/tut1
% mkdir -p dirB/dirB_1
% mkdir -p dirB/dirB_2
% mkdir -p dirC/dirC_1
% cd ~/ece4750/tut1
% 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.

2.5. 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
% cp ece4750-tut1.txt ece4750-tut1-b.txt
% ls -1
```

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.

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

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 3.3 for a useful command that you might want to use instead of the `rm` command to avoid accidentally deleting important work.

```
% 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.

2.6. 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
```

2.7. 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 laboratory assignments. The command takes the pattern and the files to search as command line arguments. The following command searches for the word "computer" in the `overview.txt` file downloaded in the previous section.

```
% cd ~/ece4750/tut1
% grep "computer" overview.txt
```

You should see just the lines within the `overview.txt` file that contain the word "computer". 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 "computer" 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 "computer".

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

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.

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

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

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

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

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

★ *To-Do On Your Own:* Try using `grep` to search for the words "computer" and "systems" in the `overview.txt` file.

2.8. 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 laboratory 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.

```
% 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
% 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.

```
% 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"`.

2.9. 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.

```
% 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.

```
% 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.

2.10. 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 out and then back into the `ecelinux` servers to see if you get assigned to a different server in the cluster.

2.11. 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/netid
% cd ${TUT1}
% pwd
/home/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.

2.12. 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.

2.13. 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.

2.14. 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.

2.15. 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 login to 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.

```
% echo "alias ct=\"cd ${HOME}/ece4750/tut1\"" >> ${HOME}/.bashrc
```

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
```

You can pipe the output of the `history` command to the `grep` command to see how you might have done something in the past.

```
% 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.

3. Course-Specific Linux Commands

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

3.1. Course Setup Script

As discussed in the remote access tutorial, the very first thing you need to do after logging into the `ecelinux` servers is source the course setup script. This will ensure your environment is setup with everything you need for working on the laboratory assignments. Enter the following command on the command line:

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

You should now see `ECE 4750` in your prompt which means your environment is setup for the course.

It can be tedious to always remember to source the course setup script. You can also use auto setup which will automatically source the course setup for you when you open a terminal. Note that if the environment for ECE 4750 conflicts with the environment required by a different course then you will need to manually source the setup script when you are working on this course. Enter the following command on the command line to use auto setup:

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

Now close the terminal and log out completely from the `ecelinux` servers. Log back in and you should see `ECE 4750` in the prompt meaning your environment is automatically setup for the course. If at anytime you need to disable auto setup you can use the following command:

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

Again, if for any reason running the setup script prevents you from using tools for another course, you cannot use the auto setup. You will need to run the setup script manually every time you want to work on this course.

3.2. Using `quota` to Check Your Space Usage

Students are allocated 10GB of storage on the servers. You can use the following command to show much space you are using:

```
% quota
```

The `blocks` column is how much data you are using, and the `quota` column is your quota. If you have exceed the 10GB quota, you can browse your home directory and list the size of files and the contents of directories with the `du` command:

```
% 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. We can pipe the output of `du` to the `sort` and `head` commands to find the top 20 largest files and directories like this:

```
% cd ${HOME}
% du -xak . | sort -nr | head --lines=20
```

Or just use the following to generate a human readable summary of the size of files/directories in the current working directory. Note that it can take 20–30 seconds for this command to finish, so please be patient.

3.3. 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.

4. Conclusion

This tutorial hopefully helped you become familiar with Linux and how to use it for working on the labs. 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 2400 Computer Systems Programming, ECE 4750 Computer Architecture, and ECE 5745 Complex Digital ASIC Design courses at Cornell University by Shreesha Srinath, Christopher Torng, and Christopher Batten.