**COEN 174 Software Engineering Fall 2021**

**Lab 1 (100 pts)**

In this lab assignment, you will learn to use Linux Bash commands and using pipes and filters to help automate commonly done tasks.

**Objective**:

* To review the commonly used bash commands
* To learn the usage of pipes and filters

**References**

A Free book – [The Linux Command Line](http://linuxcommand.org/tlcl.php)

[Unix/Linux Command reference](https://files.fosswire.com/2007/08/fwunixref.pdf)

[Wildcards](https://linuxhint.com/bash_wildcard_tutorial/)

A Tutorial at [TutorialsPoint](https://www.tutorialspoint.com/unix/index.htm)

[Manipulating files](http://linuxcommand.org/lc3_lts0050.php)

**Connecting to Linux machines in the Computing Center**

This lab will require that you work on the Linux machines. You will remotely connect to the machines using your username and password.

**Using the basic commands to find out general information**

Once you have successfully logged in, you will see the **shell** prompt. Shell is the program that acts as the user interface to read and interpret your commands and access the services of the kernel. The shell is not only a command interpreter but also supports a programming language to write programs called Shell Scripts**.**

There are a variety of shell families, each with using their own programming language. For example, the Bourne Shell family includes sh,ksh and bash. The C-Shell family includes csh and tcsh. In this lab, we will use commands for bash.

When you login, you will be in a default shell. To find out, what shell you are in, type

**echo $SHELL** at the prompt**.** If it is not bash, you can change to bash by typing **bash** at the prompt.

**How to capture the output on the console in a file for submission**

To capture the session (everything that you type and the output) on your terminal, use the **script** command. At the prompt, type **script filename***.* Give a filename of your choice. This is a text file that you will submit for this lab. From now on, everything that happens at the terminal will be captured in the filename. To end the session, type exit.

**Note:** Before you capture the output using script command, make sure that your solutions for the exercises are all working correctly. Then start your session using the **script** command and issue the command(s) for your exercises, capture the session in the file that you will submit for the lab.

**Using basic commands**

Most of you are probably familiar with the basic unix/linux commands to list files (**ls**), create a directory (**mkdir**), change permissions (**chmod**), move files (**mv**), copy files (**cp**), list file contents (**cat**) etc. If not, there are a list of basic commands and references for you to be familiar with before we move on to exercises in this lab.

Important: If you want to know how a Linux command works, type **man *commandname*** to see the man page for the command.

Read the reference to understand the command syntax (options etc) in Linux.

**Standard Input, Standard Output and Standard Error**

When you run a command in the terminal, bash creates three data streams, namely, *stdin*, *stderr*, and *stdout*. This is based on the idea that any text-based program should be able to accept input from any source and write output to any target. The source of input and the target of output are not specified by the programmer. For example, you are using a command that sorts lines of text. You can type the text at the keyboard, read it from a file, or even use the output from another program. The output can be displayed on the screen, redirected to a file or sent to another program as input.

**Reference**: [IO Redirection](https://www.tutorialspoint.com/unix/unix-io-redirections.htm)

**Part 1 (25 pts)**

**References:**

[**BASH Wildcards**](https://ss64.com/bash/syntax-wildcards.html)

Create a directory called **lab1exer1**, using **mkdir** command. To run your exercises, you will need some files in your directory. A quick way to create files is either using **touch** (empty files) or using **cat** command.

1. Create a few files, some with a **.c**, with a **.C**, a .p, **.py**, **.txt** and **.txt** extensions.

Ex:

one.c

one.C

two.py

three.py

demo1.txt

demo2.txt

demo1.cpp

prog1.sh

ex.p

Do an **ls** and check that the files are created.

Using **ls** and **wildcards** write the commands to do the following:

1. Show all filenames with two character extension **ls**
2. Show all filenames with either a c or C extension **ls**
3. Write a command to rename demo1.txt to demo1.doc. This can be done easily with a mv command. **mv**
4. Write a command to move **all** .txt files into their corresponding .doc files. For example, if you have files demo1.txt and demo2.txt in your directory, they should be moved to demo1.doc and demo2.doc respectively. You may not be able to this with **mv**.

Hint: Think of command, “rename”

[How to rename a group of files](https://www.networkworld.com/article/3433865/how-to-rename-a-group-of-files-on-linux.html#:~:text=To%20rename%20a%20group%20of%20files%20with%20a%20single%20command,be%20made%20before%20making%20them.&text=For%20decades%2C%20Linux%20users%20have,does%20just%20what%20you%20expect.)

**Pipes and Filters**

**What are filters?**

A **filter** is a command that takes the standard input (the input can come from a file or output from another program), does something useful with it, and then returns it as a standard output (can be redirected to a file or as input to another program). The commonly used Linux commands, **cat, grep** and **wc** are filters. Linux has a large number of filters. We will look at a number of them. You can also write your own filters.

**Note: We will study pipes and filters as an architectural style in weeks 6-7 of lectures.**

**What are pipes?**

In Linux, a series of filters can be used in sequence to create a **pipeline**, thus making it easy to create new and powerful programs using existing filters.

Example: cat file1 | grep “coen174” | wc -l

The programs **cat, grep** and **wc** are **filters.**

The shell connects the standard output of **cat** to the standard input of grep and the standard output of grep to wc.

**So, what does this pipelined command do?**

**cat** displays the file contents on the standard out (by default) which are now piped to **grep** command which searches for the string “COEN174” in the input (piped from cat) and pipes the output to **wc** command which with a -l option, counts the no. of lines of input.

So, the above piped command counts the number of lines that contain the string, “COEN174” in file1.

We will do a few practice exercises using the commonly used filters, grep, cat, cut, paste,sort,tr,tac and rev.

Then you will write a few pipelined commands to do the tasks given in Part 3.

**Part 2 (50 pts)**

**References**

[Grep and Regex](https://ryanstutorials.net/linuxtutorial/grep.php)

[Grep – A quick reference](https://ryanstutorials.net/linuxtutorial/cheatsheetgrep.php)

**For each of the exercises below, fill in the blank to complete the command and test it on your input files.**

**Input files**: You are given five text files in the zipped file (**inputfiles**) to use as input for testing your commands.

**Note:** Make sure you know the anchor characters ^ and $ to indicate the beginning and end of the line. What is **^ inside []** ?

Note: Regex is short for Regular Expressions.

1. Find the lines that start with a number in ditty.txt. Fill in the regular expression and run the command.

**grep ---------- ditty.txt**

1. Find the lines that do not start with a number in ditty.txt. Fill in the regular expression and run the command.

**grep ------------ ditty.txt**

1. Find the lines that have only numbers. Fill in the regular expression and run the command.

**grep -E ------------ ditty.txt**

The -E option allows extended regex to be used.

1. Find lines that either start with a number, or end with a number, but do not start and end with numbers.

**grep -E ------------- | ------------ ditty.txt**

**Note:** You must give a regex for lines starting but not ending with a number and a regex for lines ending but not starting with a number.

1. Print the lines in file, **data.txt** in reverse (with last line first) and write the output to file **tacdata.txt.**

**tac data.txt -------------**

**Reference for** [**Cut**](https://alvinalexander.com/linux-unix/linux-cut-command-field-number-delimiter-example/)

1. Examine **allstaff.txt** file. Display only the names of the staff members in the file. You must specify the field numbers with a **-f** option and the field separator with a **-d** option.

**cut -------- allstaff.txt**

1. Write the **names** and **salaries** in allstaff.txt into file, names\_salaries.txt

**cut --------- allstaff.txt**

**Reference for** [**tr**](https://linuxhint.com/bash_tr_command/#:~:text=tr%20command%20can%20be%20used%20with%20%2Dc%20option%20to%20replace,replace%20them%20by%20'a'.)

1. Change the current field separator, comma into a colon in **names\_salaries.txt** and write the output to **names\_salaries2.txt**

**tr --------- < names\_salaries.txt --------**

1. Remove (use a -d option) the “$” from the salary field in names\_salaries.txt and display the results.

**tr < names\_salaries.txt**

1. Reverse each line in **data.txt** (run the command and remember what it does. It may come in handy in later exercises).

**rev data.txt**

**Reference** [**paste**](https://en.wikipedia.org/wiki/Paste_(Unix))

1. Paste pairs of name and city from the files, **empnames.txt** and **cities.txt** and write the output to empcities.txt.

**paste empnames.txt cities.txt ----------**

**Part 3 (25 pts)**

**You will use pipes and filters to do these two exercises.**

1. List all the files that begin with a lowercase letter and don’t end with a digit.

You will pipe the output of ls into grep and use the appropriate regular expression in grep to do the task.

**ls | grep ------------------**

1. List all the files with names ending with digits 1 to 22. For example, your command should be able to list file names like file1 to file22, but not file23 and other numbers.

**ls | grep ------------------**

To test your command, you may have to create files (using touch),file1 to file25. Instead of typing the touch command multiple times on the command line, you may want to include all the touch (touch file1, for example) in a file called createfiles.sh and run it using **bash createfiles.sh.**

**Hint:** You may have to use the alternate character **(|)** in grep to achieve this task.