CS333 Laboratory Exercise #1

1 Introduction

This exercise is designed to familiarize you with man pages, access permissions, and use of shell utilities including pipes, redirection of output, shell scripts, and grep.

2 Instructions for Submitting Lab 1 Problems

You will turn in four problems for Lab 1. ***Submit each problem as soon as you complete it.*** For each problem, you will submit one file. Follow these instructions for each file that you submit:

a. Name the file properly:

*lastname_firstname*_lab1_sec*sec#*_p*prob#*_sub*sub#*.script

Where:

- *lastname* is YOUR LAST NAME
- *firstname* is YOUR FIRST NAME
- *sec#* is YOUR lab section number. Valid values are: 1 or 2 or 3
- *prob#* is the problem number. Valid values are: 1 or 2 or 3 or 4
- *sub#* is the submission count for the problem, e.g., on your first submission, use 1; if you submit the problem a second time, use 2; if you submit the problem a tenth time, use 10.

example: If student Foghorn Leghorn is registered for section 2 and submits problem 3 twice, the filename for the second submission of that problem should be:

leghorn_foghorn_lab1_sec2_p3_sub2.script

b. Attach the file to a new email message
c. Enter the name of your file as the SUBJECT of the email
d. Email your properly named file to cs333acc@cs.pdx.edu

** you can access your MCECS webmail from https://webmail.cecs.pdx.edu

3 Materials and Examples for Lab Exercises

The materials and additional examples for this lab are available at:

```
/u/karavan/public/LAB_01-1/
```

You should be able to list the contents of this directory. *Copy the contents over to your working area*.

4 Introduction to Topics Covered in this Lab

4.1 Reading the Manual

MAN pages are the MANUAL for linux/unix. When you install the OS on your workstation, the correct set of man pages for your specific OS version are installed locally. Looking up man pages on the web may yield incorrect results, that is, you may be reading a man page for an OS other than your own. To read a man page:

```
EXAMPLE> man cat
```

To learn more about man:

```
EXAMPLE> man man
```

Useful MAN pages for this lab: `cat, cd, chmod, cp, diff, file, grep, ls, mkdir, more, script, sed, tee, tr, wc.`

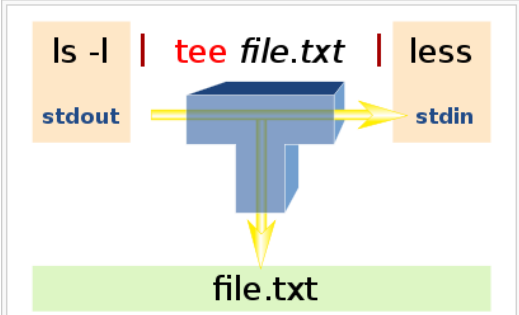
MAN page navigation:

Forward:	space, enter, j
Backwards:	b,k
exit:	q

There are not always good "usage" examples included in a man page, e.g., the man page for tee does not contain an example. Sometimes you can find a better description and examples from another source. However, *you must exercise caution* when testing information provided by a source external to your system's man pages. For example, this description of tee and the diagram may help to clarify how to use tee in Problem 2 of this lab:

tee *is normally used to split the output of a program so that it can be seen on the display and also be saved in a file. The command can also be used to capture intermediate output before the data is altered by another command or program. The tee command reads standard input, then writes its content to standard output and simultaneously copies it into the specified file(s) or variables.*

quote and image from:
en.wikipedia.org/wiki/Tee_(command)



The usage of *tee*: The output of `ls -l` is redirected to *tee* which copies them to the file *file.txt* and to the pager `less`. The name *tee* comes from this scheme - it looks like the capital letter *T*

4.2 The Shell

The command line prompt is printed by a program called the Shell. This program prints the prompt, reads your input, executes the requested commands, and displays results or error messages as appropriate. There are different versions of the shell available. In this laboratory we will use the bash shell.

To see all available shells:

```
EXAMPLE> cat /etc/shells
```

To execute a particular shell, enter its name as any other command:

```
EXAMPLE> /bin/bash
```

To return to your original shell (afterwards, you should see the original prompt):

```
EXAMPLE> exit
```

To change your shell for all future sessions (you will be asked to enter your password):

```
EXAMPLE> chsh -s /bin/bash myusername
```

4.3 Pipes

Inserting a pipe between 2 commands connects them in a particular way. The output generated by the first command is used as input to the second command, in place of keyboard input; the output of the first command does not go to stdout. (stdout is set by default to the screen).

```
EXAMPLE> ls -l | more
```

4.4 Output and Input Redirection

You can *redirect the output of a command to a file* with the greater-than symbol '>':

```
EXAMPLE> ls -l > myfile.txt

EXAMPLE> who > myfile.txt
```

In the above example, the file myfile.txt is overwritten each time we redirect to it. If you want *to append to an existing file*, use >>:

```
EXAMPLE> ls -l > myfile2.txt

EXAMPLE> who >> myfile2.txt
```

You can *redirect the input of a command to read from a file* with the less-than symbol '<':

```
EXAMPLE> wc  < myfile2.txt
```

4.5 Filename Expansion with Wildcards

Given a wildcard in a filename, the shell will expand the name appropriately. Notice, the expansion may result in zero, one, or multiple filenames.

The asterisk (*) character is used to match against any number of characters in a file name (including the number 0). To list all files with names starting with "my":

```
EXAMPLE> ls -l my*
```

The question mark (?) character is used to match against any single character. To list all files with names starting with "my" and followed by exactly four characters then ".txt":

```
EXAMPLE> ls -l my????.txt
```

4.6 Editors

There are many options for Text Editors. You may be familiar with a few:

vi, vim, gvim, emacs, pico, gedit, eclispe, ...

Some VI tips:

VI modes: *command*, *edi*t

To enter *command* mode from *edit* mode: [esc]

To enter *edit* mode from *command* mode: i

What mode are you in when you first open a file? *command*

To navigate a document while in *command* mode: h,j,k,l

The following save and quit commands must be done from *command* mode:

To save a file:	[SHIFTT]+[:]w
To quit:	[SHIFTT]+[:]q
To save and quit:	[SHIFTT]+[:]wq
To force quit without save:	[SHIFTT]+[:]q!

*** In VI avoid turning on CAPSLOCK (can be dangerous if you forget to turn it off when you switch to command mode).

Editor References:

Vi and Vim:	www.linuxconfig.org/Vim_Tutorial
Eclispse:	www.eclipse.org
GNU Emacs:	www.gnu.org/software/emacs/manual/html_node/emacs/index.html

4.7 Shell Scripts

A shell script is a program for an operating system shell to run. It is not compiled. The shell executes the program line by line. The first line of a shell script is very important – it specifies which shell to use:

```
#!/bin/bash
```

Comments: prepend the line with # (notice the #! is not a comment)

An Example Shell Script in a file named firstTest.sh (in the lab materials):

```
#!/bin/bash
echo "My First bash script"
echo "Creating and setting a local variable a to 1"
a=1
echo "Now I'm going to use the variable a ..."
echo "Var a has value" $a
```

To run this script, first change the permissions on the file, so that the owner has 'execute' permissions:

```
EXAMPLE> ./firstTest.sh
```

4.8 sed

sed is a stream editor for filtering and transforming text. The man page for sed is pretty good. A tutorial for sed, with many good examples, can be found at www.sedtutorial.com. You can try the following sed examples using the `sed_example.txt` file from your lab materials:

Show input file:

```
EXAMPLE> cat sed_example.txt
one   1
two   1
three 1
one   1
two   1
two   1
three 1
```

If 'two' is found on a line, replace any '1' that follows on the line with '2':

```
EXAMPLE> sed '/two/ s/1/2/' sed_example.txt
one   1
two   2
three 1
one   1
two   2
two   2
three 1
```

Show input file:

```
EXAMPLE> cat sed_example.txt
```

Connect multiple sed commands in a single line:

```
EXAMPLE> sed '/two/ s/1/2/; /three/ s/1/3/' sed_example.txt
```

Show input file:

```
EXAMPLE> cat sed_example.txt
```

Save the output from sed by redirecting it to a file:

```
EXAMPLE> sed '/two/ s/1/2/; /three/ s/1/3/' sed_example.txt > out.txt
```

Compare the saved output file to the input file:

```
EXAMPLE> diff sed_example.txt out.txt
2,3c2,3
```

```
< two   1
< three 1
---
> two   2
> three 3
5,7c5,7
< two   1
< two   1
< three 1
---
> two   2
> two   2
> three 3
```

4.9 Regular Expressions

Regular expressions are used to match strings of text; they are especially powerful when searching for patterns in text. In this lab, you will use grep, a Linux/Unix utility, to search for patterns in text. grep is a regular expression processor, and given a properly formed regular expression, grep searches files for the pattern specified in the regular expression. The man page for grep is useful for showing the general syntax of a grep command, but it falls short in explaining and describing the regular expression language it expects. There are many tutorials on the Internet about regular expressions and grep. The following grep examples work with `phonenumbers.txt`, which is provided as part of the lab materials:

Find all numbers that start with 2:

```
grep '^[2]' phonenumbers.txt
```

Find all numbers of exactly length 10:

```
grep '\<[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]\>' phonenumbers.txt
grep '\<[0-9]\{10\}\>' phonenumbers.txt
```

Find all numbers of at least length 10:

```
grep '\<[0-9]\{10,\}\>' phonenumbers.tx
```

Find all numbers that start with 5 and contain pattern 511:

```
grep '^[5][0-9]*511[0-9]*' phonenumbers.txt
grep '\<[5][0-9]*511[0-9]*\>' phonenumbers.txt
grep '^[5][0-9]*511[0-9]*\>' phonenumbers.txt
```

Let's say that 0 and 1 are *special* digits. Find all numbers that contain at least four *special* digits:

```
grep '\<[2-9]*[01][2-9]*[01][2-9]*[01][2-9]*[01][2-9]*\>' phonenumbers.txt
```

4.10 SCRIPT (the utility, which is different from a "shell script")

Look over the man page for the SCRIPT utility. You will use SCRIPT to *record your work* in this lab.

5 Grading

Turn in 4 SCRIPT files (recordings from the SCRIPT utility), one for each of the 4 problems. A good approach is, for each problem: first read the man pages, then figure out (by experimenting) what the one-line command sequences should be, then use SCRIPT to record your solution for the submission.

Submit your completed work as instructed in section 2 of this document.

Late Submissions: A problem is late if it is received after your lab section ends. *Late submissions for this lab are accepted up to four days after the start of your lab section. For example, Wed night students must submit all lab problems by 6:40pm on Sunday.*

Submit late problems by email, following the instruction given in section 2 of this document.

6 Problem Set

Problem 1: Use SCRIPT to record steps (a) - (k) below. Each step should be done with just one command line.

a. Make a directory named `Lab1InFiles`
b. Change directories so that you are in `Lab1InFiles`
c. List the contents of `Lab1InFiles`
d. Copy all files ending with ".in" from the Lab 1 materials directory to your `Lab1InFiles` directory
e. List all the files in `Lab1InFiles` and show the access permissions
f. Change the access permissions on all of the files in `Lab1InFiles` so that the user has rwx access permissions, but group and other have no access permissions
g. List all the files in `Lab1InFiles` and show the access permissions
h. Display the contents of `Lab1File1.in`
i. Display the contents of `Lab1File2.in`
j. Execute this command line: `./Lab1File3.in Lab1File1.in Lab1File2.in`

   *** Notice that `Lab1File3.in` is a shell script that takes 2 command line arguments. (The shell script name is misleading; we have done this to show that the name is not what determines what is executable.)
k. List the contents of `Lab1InFiles` and show the access permissions.

Problem 2: First, create the command line for part A and the shell script for part B. Then, use SCRIPT to record steps i-v in (a) and (b) below.

a. Use `file`, `grep`, `tee`, `wc`, pipes, and wildcards to generate a file named `dumpFile1` that contains a line of information from the `file` utility for each ascii file in `Lab1InFiles` and to print to stdout the number of ASCII files found. *Do this in one command line*.

   i. Change directories so that you are in `Lab1InFiles`

   ii. List the contents of `Lab1InFiles`

   iii. Execute your one-line command

   iv. Display the contents of `dumpFile1`

   v. List the contents of `Lab1InFiles`

b. A shell script is a file containing Linux commands. Construct a shell script named `findDir` with this functionality: `findDir` should store the information from the `file` utility for each ascii file in a directory into a file named `dumpFile2`. In addition, the number of ASCII files should be stored at the end of `dumpFile2`, rather than printed to stdout as you did in part (a). `findDir` must be executable.

   i. Display the `findDir` shell script

   ii. Execute `findDir`

   iii. Display the contents of `dumpFile2`

   iv. List the files in `Lab1InFiles` and show the permissions

   v. Use a Linux command to show that `dumpFile1` and `dumpFile2` are not the same.

Problem 3: Use SCRIPT to record steps (a) - (c) below.

Look up `tr` and `sed` in the man pages. Use these in a shell script that you write, named `p3.sh`. Your shell script should take one command line argument (which will be a filename), and it should have this functionality: convert uppercase to lowercase; convert lowercase to uppercase, and convert all blanks to hyphens; and copies all other characters with no change. Inside a shell script, command line arguments are referenced by their position. So you can refer to $1, $2, etc. See `Lab1File3.in` for an example.

a. Display your shell script

b. Execute your shell script on `prob3.test1` and redirect the output to `p3.out`

c. Display `p3.out`

Problem 4: Use SCRIPT to record steps (a) - (b) or (a) – (c) below.

Use grep to locate the patterns below in the file `/usr/share/dict/words`. Limit your matches to lowercase letters.

a. All words that start with the letter c or d that also contain the pattern uou. Example: continuous

b. All words of length 5 that contain no vowels (vowels are a, e, i, o, and u).* Example: gypsy.

c. *Challenge Problem (part c is optional)*. If you do part c, include it in your recorded SCRIPT: All words containing at least 7 vowels.* Example: audiovisual.

* If you can't figure out how to strip away the apostrophe s, that is ok.