CS201 Hands-On Laboratory Exercise #1

This exercise is designed to familiarize you with the script utility, man pages, access permissions, and use of shell utilities including pipes and redirection of output. You will also work with gcc and the vim editor.

Part 1: Required Problems

Problem 1: Using the Linux Shell
Use the `script` utility to record steps (1) - (11) below. Each step should be done with just one command line. Submit the file created by script.

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

1. Make a directory named Lab1Files
2. Change directories so that you are in Lab1Files
3. List the contents of Lab1Files
4. Copy all files from /u/karavan/public/LAB_01-1/ to your Lab1Files directory
5. List all the files in Lab1Files and show the access permissions
6. Change the access permissions on all of the files in Lab1Files so that the user has rwx access permissions, but group and other have no access permissions
7. List all the files in Lab1Files and show the access permissions
8. Display the contents of Lab1File1.in
9. Display the contents of Lab1File2.in
10. Execute this command line:
    `./changeIt Lab1File1.in Lab1File2.in`
    `changeIt` is a shell script that takes 2 command line arguments. Note the file permissions for `changeIt` must include permission to execute.
11. List the contents of Lab1Files and show the access permissions.

Problem 2: Hello World (hello.c)

Compile the program hello.c using gcc. You will see an error message. Fix the error and compile again.

Use a compiler option to name the executable program "hello". [hint try man gcc to see the common options]

Copy hello.c to a new file, alien.c Edit alien.c to print out "hello, world 2" instead of "hello, world". [hint: try man printf to see the format string information you need for this change]

Problem 3: Math (tan.c)

File tan.c contains a not-quite-right version of a program to print out the tangent of 90. First, try to compile it with gcc. Then, fix the errors so that it compiles and runs correctly. [hint: try man tan]

Problem 4: assembly language and object file

Use gcc to generate assembly language and an object file for your working version of hello.c . You will need options -S and -o. Take a look at each of these files. [instead of the editor, it will be easier to use more or less.] Now check the file sizes. Which is larger?

Part 2: [optional] Work through as many as time allows

Problem 5

First, create the command line. Then, use SCRIPT to record steps i-v.

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 Lab1Files 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 Lab1Files

ii. List the contents of Lab1Files

iii. Execute your one-line command

iv. Display the contents of dumpFile1

v. List the contents of Lab1Files

Problem 6

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.

1 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_ pprob#.script Where:

- lastname is YOUR LAST NAME
- firstname is YOUR FIRST NAME
- prob# is the problem number. Valid values are: 1 or 2 or 3 or 4

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 cs201acc@cs.pdx.edu
[NOTE: you can access your MCECS webmail from https://webmail.cecs.pdx.edu

2. 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.

4 Introduction to Topics Covered in this Lab

4.1 MAN: 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, tee, tr, wc.

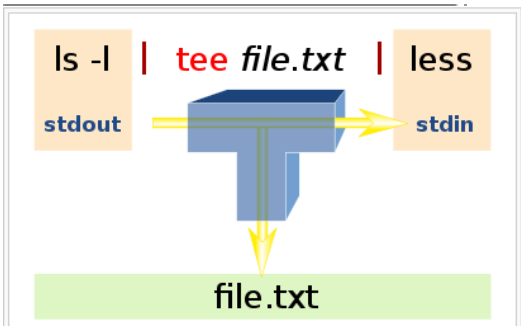
MAN page navigation:

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, but BE SURE IT IS THE SAME SYSTEM VERSION YOU ARE USING.



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*

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)*]

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, edit To enter command mode from edit mode: To enter edit mode from command mode:

[esc] i

What mode are you in when you first open a file? To navigate a document while in command mode: The following save and quit commands must be done from command mode:

To save a file: To quit: To save and quit: To force quit without save:

[SHIFTT]+[:]w [SHIFTT]+[:]q [SHIFTT]+[:]wq [SHIFTT]+[:]q!

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

4.7 Shell Scripts

www.linuxconfig.org/Vim_Tutorial www.eclipse.org
www.gnu.org/software/emacs/manual/html_node/emacs/index.html

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 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