Cabrillo College

Final Project

For the final project you will be writing custom front-ends to your favorite Linux commands. To do this you will write a shell script that interacts with the user to get input, then use that input to call a Linux command. You will start with a template that you can modify and extend.

Forum

Use the forum to brainstorm script ideas, clarify requirements, and get help if you are stuck. When you have tested your script and think it is bug free then use the forum to ask others to test it some more. Post any valuable tips or lessons learned as well. Forum is at: http://oslab.cishawks.net/forum/

Commands

.	echo	lpstat	sort
at	env	ls	spell
banner	exit	mail	su
bash	export	man	tail
bc	file	mesg	tee
cal	find	mkdir	touch
cancel	finger	more	type
cat	grep	mv	umask
cd	head	passwd	uname
chgrp	history	ps	unset
chmod	id	pwd	vi
chown	jobs	rm	wc
clear	kill	rmdir	who
cp	ln	set	write
date	lp/lpr	sleep	xxd

Commands by Categories

Type	Commands
Simple:	banner, date, cal, finger, uname, type, ls, tree, cd, pwd, hostname
Status:	jobs, ps, who, id, env, lpstat, umask, mesg, history, tty
File:	file, cp, mv, ln, rm, rmdir, mkdir, touch, chmod, chgrp, xxd, head, tail
Filters:	cat, sort, spell, grep, wc, tee
Misc:	bc, mail, vi, lp, cancel, at, kill, find, passwd

The Template

```
#!/bin/bash
#
# menu: A simple menu template
#
while true
do
        clear
        echo -n "
                CIS 90 Final Project
        1) Task 1
        2) Task 2
        3) Task 3
        4) Task 4
        5) Task 5
        6) Exit

        Enter Your Choice: "
        read RESPONSE
        case $RESPONSE in
          1)    # Commands for Task 1
                ;;
          2)    # Commands for Task 2
                ;;
          3)    # Commands for Task 3
                ;;
          4)    # Commands for Task 4
                ;;
          5)    # Commands for Task 5
                ;;
          6)    exit 0
                ;;
          *)     echo "Please enter a number between 1 and 6"
                ;;
        esac
        echo -n "Hit the Enter key to return to menu "
        read dummy
done
```

Procedure

Copy the template text to a file in your own bin directory named **myscript**. Give the file execute permission for everyone. You will run your script by entering its name on the command line just as you do with any other Linux command.

Choose five commands you would like to develop a custom front-end user interface for. For each choice add a menu option and then develop the appropriate case statement. Expand the case statement with commands to query the user for input, then use that input for calling your command.

Each command front-end that you implement should include:

- User dialog to prompt the user for input
- Use variables to save the input
- Execute one or more commands using the variables as the options or arguments.

Make sure your script can run by itself and run from **allscripts** which is in the */home/cis90/bin* directory. You also need to set permissions so that everyone in the cis90 group can read and execute your script.

When finished, you will need to test your script and repair any defects you find. After your own testing it is helpful to have others test your work. Developers can't always imagine all the creative ways users will use their products. Use the forum to ask other students to run your script and give your feedback. This is also a good way to check your script can be run by others in the cis90 group.

You can use any development method you wish. With the waterfall method you start by identifying high level requirements, then write a functional specification, then do a design, then code, then do unit testing of individual pieces, then do system testing of the whole product. The idea is to delay coding until you have thought everything through first. With methods like evolutionary delivery you start small with a working initial version of the project and then do rapid evolutions of mini-waterfalls (design-code-test) to iteratively add new functionality. Another method is to do rapid prototyping to flesh out the design, and then use a waterfall method to build it. The evolutionary method would be ideal for a small project like this. Good old "hacking" is a blend of rapid-prototyping and evolutionary delivery.

Tips

- Run **allscripts** to see some examples of what Homer and Benji developed.
- Homer's work on the front-end to the **grep** command is a good example of what is expected. However Homer has more work to do when he gets back from playing fetch as not all five tasks are completed.
- Benji went a tad overboard on his tasks. He must been thinking he would get chicken treats. You don't have to do as much as Benji did but you might get some ideas from viewing his work. His script also has many examples of how to do conditionals for other students wanting to go over the top.
- Use **vi** to look at Homer and Benji's scripts in their bin directories. **vi** adds color to make the commands easier to read.

- Save earlier versions of your work. You can easily do this by copying **myscript** to *myscript.v1*, *myscript.v2*, etc. to backup each version.

Shell scripting topic ideas

An easier to use find command

View the File System
- Show files at the root (/) directory
- Show files in user's home directory
- Show files in the parent directory
- Show files in a specified directory

Tally Files
- Tally the number of directories from a given starting point
- Tally the number of symbolic links from a given starting point
- Tally the number of text files from a given starting point
- Tally the number of shell scripts from a given starting point
- Tally the number of data files from a given starting point
- Tally the number of symbolic links from a given starting point
- Draw a histogram showing the relative numbers of different file types

Search for files
- Search for a file by name
- Search for a file by inode number
- Search for a file by owner
- Search for a file by size
- Search for files that have been modified recently
- Search for files that contain a certain string of characters

View files
- Select a file for viewing
- View the entire file allowing paging through the file
- View a file in hexadecimal format
- View the top 20 lines of a file
- View the bottom 20 lines of a file

Process status
- Show just system processes
- Show processes belonging to a particular user
- Show any defunct processes
- Show all processes

To turn in

Copy your final version of **myscript** as follows:

```
cp myscript /home/rsimms/turnin/cis90/myscript.$LOGNAME
```

Grading rubric (60 points maximum)

Possible Points	Requirements
30	Implementing all five tasks (6 points each): • Requirements for each task: – Minimum of 10 "original" script command lines – Has one or more non-generic comments to explain what it is doing – Has user interaction
25	You don't have to do all of these but do at least five: • Redirecting stdin (5 points) • Redirecting stdout (5 points) • Redirecting stderr (5 points) • Use of permissions (5 points) • Use of filename expansion characters (5 points) • Use of absolute path (5 points) • Use of relative path (5 points) • Use of a PID (5 points) • Use of inodes (5 points) • Use of links (5 points) • Use of scheduling (5 points) • Use of a GID or group (5 points) • Use of a UID or user (5 points) • Use of a /dev/tty device (5 points) • Use of a signal (5 points) • Use of piping (5 points) • Use of an environment variable (5 points) • Use of /bin/mail (5 points) • Use of a conditional (5 points) The maximum for this section is 25 points.
5	Present your script to the class
Points lost	
-15	Fails to run from **allscripts**
-15	Other students in the class are unable to read and execute your script.
-15	Error messages are displayed when running one or more tasks
-up to 90	No credit for any task which contains unoriginal script code that: • Doesn't give full credit to the original author • Doesn't indicate where the code was obtained from • Doesn't include licensing terms • Violates copyright or licensing terms
Extra credit	
30	Up to three additional tasks (10 points each)