Assignment 4: Due March 22nd in class

Basic Information

Unix/Linux Scripting Quick reference

Task/Concept	**Example**
Starting the script: #![path to shell you want to use]	#!/bin/bash

Variables:
Assignment: variablename=
usage: $variablename
	number=5
	somestring="here's your beef!"
	
	echo "Oh yeah? Well $somestring"

Program Flow:
if [conditional]
then
fi
 	
if [conditional]
then
...
else
...
fi
	if [$number -gt 4]
	then
	  echo "Your number is greater than 4"
	fi
	
	if [$number == 7]
	then
	  echo "Your number is seven"
	else
	  echo "Your number is stupid"
	fi

Conditional(arithmetic): -eq == -ne != -lt < -le <= -gt > -ge >= **Conditional(string):** = != -n (string is not null and DOES exist) -z (string exists but is null)	[$number -eq 5] (test whether value of number = 5) [string1 = string2] (test if string1 = string2) [-n string1] (test if string1 has data)

Reading from a file:
cat filename | while read variablename
do
...
done
 	
or
 	
while read variablename
do
...
done < /directory/filename
	cat /etc/passwd | while read lineoffile
	do
	/*manipulate $lineoffile somehow */
	done
	
	while read lineoffile
	do
	/*manipulate*/
	done < /etc/passwd

Reading from the keyboard read variablename	read n (fill n with user input)
Piping & redirection: program1 | program2 *send output of 1 to 2* program1 > filename *put output of 1 in filename* program1 >& filename *put output&errors in filename*	Ps -ef | grep root *(shows root processes only)* crontab -l > cron *(puts the cron table into a file)* somescript.sh >& /dev/null *(runs a script without any kind of visible output)*

Program Structure

The program is going to be menu driven. The user should have the 4 choices presented to them. The menu will be driven by a while statement containing a case statement.

- `while` statement

  Syntax:

  ```
  while [ condition ]
  do
    command1
    command2
    command3
    ...
    ...
  done
  ```

  Loop is executed as long as given condition is true. Make sure to surround the brackets with spaces (the shell parser requires it). The condition itself can be formulated using logical expressions.

- `case` statement

  Syntax:

```
case  $variableName  in
  pattern1)
    command
    command
    ...
    command
    ;;

  pattern2)
    command
    command
    ...
    command
    ;;

  ...
  patternN)
    command
    command
    ...
    command
    ;;

  *)
    command
    command
    ...
    command
    ;;
esac
```

  The case statement is like a C++ switch case statement. The variable *$variableName* is compared against the specified patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and its executed if no match is found.

Option 1: Print accounts registered on the machine

This option uses the /etc/passwd file. This file contains basic user attributes, it is an ASCII file that contains an entry for each user. Each entry defines the basic attributes applied to a user. When you use the mkuser command to add a user to your system, the command updates the /etc/passwd file.

An entry in the /etc/passwd file has the following form:

Name : *Password* : *UserID* : *PrincipleGroup* : *Gecos* : *HomeDirectory* : *Shell*

This part will use the cat command in conjunction with the while command (see the “Reading from a file” in the basic information section) and the cut command.

- `cat` command

  This command is used to output the contents of a file to standard out. For example, ‘cat /etc/passwd’ entered into the command line will output the contents of the file /etc/passwd to standard out.

- `cut` command

  The cut command is used to output a column of data. Say we have a data file called data.txt. In this file each row has 8 entries that are separated by ‘||’. If we just want to see the first entry for each row then the command we would enter would be:

  ```
  cat data.txt | cut -f1 -d"||"
  ```

  The first option (-f1) tells the cut command that you want the first field and the second option (-d”||”) defines what separates the fields (i.e., the delimiter). So this command takes the input and outputs the first field, with fields delimited by ‘||’. Say the data is now space separated and we want the first and third fields then we could do:

  ```
  cat data.txt | cut -f1,3 -d" "
  ```

Option 2: Print out the home directory and disk usage for each registered user

This will work like option 1 with additional functionality. You will get the registered users and their home directories from /etc/passwd using the cat and cut commands. Next you will output the disk usage of each file using the du and tail commands.

Careful: Some of the home directories will be null. If this is the case (think about using –z (shown in the basic information) to tell if a string is empty/null).

Careful: Some of the home directories will be the root (“/”). DO NOT du these directories (it would take forever). Make sure and check for this and report it.

Careful: You will not have permission to access some of the home directories, if this is the case then du will have an error. I don't want to see these errors. You can redirect stderr using '2>".

Ex: someCommand 2> /dev/null | anotherCommand

Here the error output of someCommand will be redirected to /dev/null (the black hole of Unix machines) and all output will redirected as input to anotherCommand.

- `du` command

  The du command outputs the disk usage for a specified directory. The syntax for the command will be:

  ```
  du -h someDir
  ```

  The –h option tells du to output the size in human readable format. If the specified directory has sub-directories then it will also output the sizes of those, the last line of output will be the total size for the specified directory.

- `tail` command

  The tail command is used to output the last line(s) of a specified input file. For example, say we did:

  ```
  ps -acefl | tail -n5
  ```

  The –n5 option tells tail to only output the last five lines of the ps command; if you wanted the last line only then you would use –n1.

Option 3: Print out the logged in users

You will need to perform a who command piped to a cut command (we don't need all of the information that is output by who, just the user name)

- `who` command

  The who command outputs the currently logged in users (along with some other information)

Option 4: Exit

Quit the program with exit.