

Instructions

Complete the problems assigned below and turn in your answers by the due time above. To receive full credit, you must show both the relevant command(s) and output. If a question asks about what a command does, or what the output would be, give both the command and the output. Demonstrate that you actually performed the command, and didn't just guess.

Homework Problems

1. Complete Exercises 3 and 4 on page 116 of your textbook. For these problems, you are to create a shell program (place the commands inside a file), and execute the shell program. Do not enter the commands on the command line. Use the template on the class website or you can copy `~cappella/files/template.sh`

Problem 3

```
#!/bin/sh

# displays number of files in current directory.
# the -l (one) option to ls means output in a single column

ls -l | wc -l
```

Now to test the program...

```
$ ./nf
1
$ touch foo
$ ./nf
2
$ ls
foo nf*
```

Problem 4

```
#!/bin/sh

# displays sorted list of logged in users
# only shows user name

who | cut -d' ' -f1 | sort
```

Testing the program...

```
$ ./whos
cappella
cappella
root
root
user1
```

The problem did not require only showing a single login name per user; if it had, you could added the `-u` option to `sort`.

2. Modify the program that you created in exercise 3 to use the `PWD` variable. `PWD` is a read-only variable set by the shell to always be your current working directory.

```
#!/bin/sh

# displays number of files in current directory
# the -l (one) option to ls means output in a single column
```

```
ls -l $PWD | wc -l
```

testing the script

```
$ cd /
$ /home/cappella/cis68b1/nf.pwd
17
$ ls
bin/   dev/  home/  lost+found/  new  orig  root/  tmp/  var/
boot/  etc/  lib/   mnt/         opt/  proc/  sbin/  usr/
```

The PWD variable really adds no value here, since **ls** uses the current working directory if no arguments are supplied. Still, PWD could be used in messages output to a user.

3. Write a shell program that gives a nice message telling the user the number of files and directories contained under his/her home directory. The program must use the variable HOME, which always contains your home directory path, and LOGNAME which is the user's login name.

```
#!/bin/sh

# displays number of files in user's home directory.
# uses $HOME and $LOGNAME

echo User: $LOGNAME
echo Home directory: $HOME
echo -n Number of files in home directory:
ls -l $HOME | wc -l
```

Testing the program from the current directory, and from the root directory to be sure the usage of \$HOME was correct.

```
$ ./nf.home
User: cappella
Home directory: /home/cappella
Number of files in home directory: 11

$ cd /
$ /home/cappella/cis68b1/nf.home
User: cappella
Home directory: /home/cappella
Number of files in home directory: 11
```

4. Create another shell program like the previous one, but this time it should count the number of files and directories under the user's entire home directory tree. Do not use the **find** program. Be sure to catch all the files, and validate your results.

This problem requires you to use the **-R** recursive option to the **ls** command. There is one little problem with that option for this problem - it causes **ls** to include some extraneous output. Here's an example (I'm using \$PWD instead of \$HOME, so that I can get less output for the example):

```
$ ls -Rl $PWD
/home/cappella/cis68b1:
dirl
foo
nf
nf.home
nf.homeall
nf.pwd
whos

/home/cappella/cis68b1/dirl:
dummy
```

Diagram illustrating the output of the recursive **ls** command. The output shows two directory listings. The first listing is for the current directory, and the second is for the subdirectory **dir1**. The output includes directory headings (e.g., **/home/cappella/cis68b1:**) and a blank line between the two listings. Arrows point from the text "Directory headings" to the directory paths in the output, and from "Blank line between listings" to the blank line between the two directory listings.

There are two problems with the output from the recursive listing. Directory headings are listed for the current directory and each subdirectory, and a blank line is placed between each sub-directory listing. To count the number of files and directories accurately, the directory headings and blank lines will have to be filtered out. Blank lines are easy to remove. You can use many tools, such as **sed** or **awk**. Since **sed** has a command to delete lines that match a pattern, we'll use it. The command to delete blank lines from output is this:

```
sed /^$/d
```

Now, to remove the directory headings - notice that they all are directories names, including slashes and end with a colon character. This is an easy pattern to match as well using **sed** and the command would be:

```
sed /\./.*:/d
```

Putting the two **sed** expressions together, and having **sed** filter the output from **ls**, we would have:

```
$ ls -Rl $PWD | sed -e '/\./.*:/d' -e '/^$/d'
dirl
foo
nf
nf.home
nf.homeall
nf.pwd
whos
dummy
```

Now we just count the lines with **wc**, and that gives us a total count of the files and directories, recursively. I've also changed the PWD into HOME, as required for the problem.

```
$ ls -Rl $HOME | sed -e '/\./.*:/d' -e '/^$/d' | wc -l
#!/bin/sh

# displays total number of files/dirs under user's home directory tree.
# uses $HOME and $LOGNAME

echo User: $LOGNAME
echo Home directory: $HOME
echo -n Total number of files/dirs under home directory tree:
ls -Rl $HOME | sed -e '/\./.*:/d' -e '/^$/d' | wc -l
```

I'll bet you forget that there are hidden dot files not listed with **ls** by default - you need to add the **-a** option to **ls** to count those too! And this creates another problem. The files **.** and **..** are listed many times, and are already counted. We really should remove those with **sed** as well, but I'll leave that to you, since it is really easy.

5. Assign a variable with a value. Now, output the value of that variable prepended with the letter X and appended with the letter Y (that is, the output should look like XvalueY).

```
$ myvar=testing
$ echo X${myvar}Y
XtestingY
```

Let's try it with a null value

```
$ myvar=
$ echo X${myvar}Y
XY
```

6. Assign the numbers 1, 2, and 3 to three variables A, B, and AB respectively. Now output the value of these variables in the order given on a single line without any spaces in between the values.

```
$ A=1
$ B=2
$ AB=3
$ echo $A$B$AB
```

Since it is difficult to tell if the last \$AB is \$A followed by the letter B, or is really \$AB, it is best to use { } to protect the variable name. A clearer solution is:

```
$ echo $A$B${AB}
123
```

7. Write a command that demonstrates and proves that a variable's value is null.

```
$ null=
$ echo x${null}y
$ xy
```

8. Using only three variables, write a command line that has two arguments (or options). Execute that command line using only the variables.

```
$ cmd=who
$ arg1=am
$ arg2=i
$ $cmd $arg1 $arg2
cappella pts/1 Jan 21 12:46 (xxx.dsl.snfc21.pacbell.net)
```

or without using spaces on the command line:

```
$ cmd=who
$ arg1=' am'
$ arg2=' i'
$ $cmd$arg1$arg2
cappella pts/1 Jan 21 12:46 (xxx.dsl.snfc21.pacbell.net)
```

9. Write a shell program that will output all the user names for users currently logged in, except that the login name of the person executing the shell program must be output in all uppercase. Make sure your program works with any number of users on the system.

This is a really easy problem, that many people make much more difficult than it is. Think of the problem in two parts - first, a list of users minus your login name, and second your own login name in uppercase. If I asked you to do each of these problems separately, you would have no trouble. But doing them together, causes brain overload! Let's avoid the overload, and do them separately.

A list of users, minus your name. Run the who command, cut out everything except the user names (first field):

```
who | cut -d' ' -f1 | grep -v $LOGNAME
```

I was surprised by the number of people who made this next step very difficult. Many of you ran who, just to get your own login name by using grep and \$LOGNAME. Since you already know the user running the program is logged in, and the name is \$LOGNAME, there is no reason to run grep. You already have what you need with \$LOGNAME. The only work required is to output \$LOGNAME in uppercase:

```
echo $LOGNAME | tr '[a-z]' '[A-Z]'
```

Now join them together in a script:

```
#!/bin/sh

# Output users on system, with current user's name in uppercase

echo $LOGNAME | tr '[a-z]' '[A-Z]'
who | cut -d' ' -f1 | grep -v $LOGNAME
```

And test the script...

```
$ ./my-users  
CAPPELLA  
zengkan1  
gever  
zengkan1  
hahnvirg  
zamaniho
```