

CIS 213: Foundations of Computer Science—Lab 1 Part II

This lab is to introduce you to the world of shell scripting. The first lab explored various unix commands. This lab will show how these commands be used as part of shell programming.

1 Shells

The shell is the command-line interpreter, where you enter your commands. The shell not only processes commands such as `ls`, `cp`, `mv`, etc., it also has a built in programming language with control strutures for selection and iteration. It also has the ability to assign variables and perform arithmetic computation. lets take a look at an example.

1.1 Begin Lab

1. `cd` to your `cis213` directory
2. make a `lab1.2` directory
3. `cd` to the `lab1.2` directory

You will be writing all of your scripts in this directory. Lets jump right into your first script.

Using your editor, open the file `display` (no file extension), and put the following text into that file:

```
#!/usr/bin/ksh

echo "The Current Date and Time: "
date
echo
echo "Your username is: `whoami` \n"
echo "Your current working directory is: \c"
pwd
```

Save the file, and at your command line prompt:

```
host% chmod u+x display
```

This makes the file executable, and now you can run it:

```
host% display
```

Open another shell window, and type:

```
host% display
```

It does not work in the other window. You should get a command not found error. This is because the `dirstats` file needs to be moved to your path.

```
host% mkdir ~/bin
```

On our systems, `~/bin` is in everyone's path. to verify:

```
host% echo $path
```

You should see `/home/<username>/bin` in the list. Now copy your script to the `~/bin` directory:

```
host% cp ~/cis213/lab1.5/display ~/bin
```

```
host% rehash
```

The `rehash` is needed to ensure that the shell will be able to find the new file. `rehash` reloads the shell's hashtable.

```
host% display
```

Now it should work :)

So you have not successfully created your first shell script. Remember the major steps:

1. Create a text file for the script
2. Place the `#!/usr/bin/ksh` at the top of the file
3. Enter the commands
4. Save the file
5. Make the file executable
6. Copy the file to your `/bin` directory

1.2 How does *display* Work?

```
#!/usr/bin/ksh
```

```
echo "The Current Date and Time: "  
date  
echo  
echo "Your username is: `whoami` \n"  
echo "Your current working directory is: \c"  
pwd
```

The commands `date`, `whoami`, and `pwd` were covered in the first lab. The `echo` displays information on stdout. The `\n` and the `\c` adds an extra carriage return and keeps information on the same line respectively.

The `whoami` is placed within the double-quotes unlike the other commands. However, it still executes. This is because it is placed with the single-back-tick quotes (NOTE: these are not single-quotes). The single-back-tick quotes are used for command substitution; this is where the result of a command is to be used for a purpose.

2 New Commands

One of the reasons for building shell scripts is to provide new commands that are more convenient. These scripts act as abstractions of frequently used operations. In this section, we will look at a shell script that represents this type.

2.1 Find-By-Name (FBN)

The Find-By-Name script will search from the current directory through all of its subdirectories to locate a file. For example, (do not try these because it does not exist yet):

```
% fbn "lab*"
Locates all of the files that start with lab
```

```
% fbn "*.cpp"
Locates all of the files that end with cpp
```

Open a file in your lab directory called *fbn* and put the following code:

```
#!/usr/bin/ksh

if test $# -eq 0
then
    echo "usage: $0 <pattern>"
    exit
else
    find . -name "$1" -print | cat -n
fi
```

This program makes use of the *test* and the *if-then-else*. The syntax is as follows:

```
test <expression>
```

where expressions can be: a -eq b, a -lt b, a -gt b, a -ne b, etc.

```
if <test>
then
    commands
else
    commands
fi
```

The program compares `$#` (a special variable to tell you how many arguments were on the command line with the script): (examples, do not type)

```
% myscript -x foo
two arguments: -x and foo
```

3 File Utilities

Many shell scripts have been written to be shell utilities. In this section we will look at 3 file utility scripts.

3.1 Directory Stats

This script is used to show the following stats for all of the files/directories in the current working directory.

```
#!/usr/bin/ksh

count=1
for f in *
do
    echo "$count --> `file $f`"
    count=`expr $count + 1`
done

echo "There are $count files"
```

Put the above script in a file called “dirstats” in your lab directory. Save the file, and follow the above to set it to be executed from within your path.

This script uses the *for* statement. The syntax:

```
for var in list-of-words
do
    commands
done
```

Therefore the program above is looping over the list of names represented by “*”. The “*” means all of the filenames in the current directory. You could also put “*.cc” or “*.cpp” or “*.txt” to restrict the list.

The “count=1” and “count=`expr \$count + 1`” are examples of arithmetic expressions and assignments. Note the use of the back-ticks again to evaluate the addition expression. The *expr* is the command, and the “\$count + 1” are the arguments. Details can be found using *man expr*.

4 Simple Contacts Book

This section show how you can build a very simple address/telephone number lookup utility. It involves a file that will contain your contacts (one entry per line), and a script that will allow you to search.

First, you will need to create a file called `.phonelist` in your home directory. The file begins with “.” to make it a hidden file. Hidden files can be seen with “ls -a”. After you have opened this file, you may fill it with the following sample information:

```
Jane Doe jdoe@spelman.edu (h) 404-555-2334
Chris Johnson johnson@somewhere.com (w) 713-555-2325
Andrea B. Johns ajohns@mit.edu (h) none (w) 812-667-9876
```

Now save this file. And in your lab diretory open a new file called *phone*. This will be the place to put the script.

place the following code in that script:

```
#!/usr/bin/ksh

if test -f $HOME/.phonelist
then
    grep -i $* $HOME/.phonelist | cat -n | less
    echo
fi
```

This script is very simple, yet can be very powerful. It can search via wildcards, case insensitive, and match any parts of words and numbers.

```
host% phone john
```

```
host% phone "An*"
```

```
host% phone 2
```

5 Exercises

You will need to consult this web page for more information in completing these exercises.

1. Alter the *display* script to show the last time you logged into the current machine. You will need to use the last command, other commands, and pipes (`|`).
2. Add prompt and loop to the phone *script* to ask the user if they would like to do another search. If they answer “y” then ask for a search value, otherwise exit. HINT: Use the *read* command to read user input. See www.mcsr.olemiss.edu/unixhelp/srpt/ for help.