Log onto your account using Putty

Create a directory called module12. All files we create will be put in there and MUST have the proper permissions for a script.

1. Use vim to create the following file called **myscript.sh**

*#!/bin/bash  
echo "Hello World"*

Save the file and use chmod to change the permission to rwx-------

What permission did you do?

chmod 700 myscript.sh

Run the script by typing: *bash myscript.sh*

2. Typically I will test out a script on the command line before writing it. So we will do that here.

Type the command that will make a directory called trash

What did you type?

mkdir trash

Type the command that will move all the files in your module12 directory into the directory trash

What did you type?

mv ~/module12/\* ~/trash

Type the command that will remove the directory and all it's contents.

What did you type?

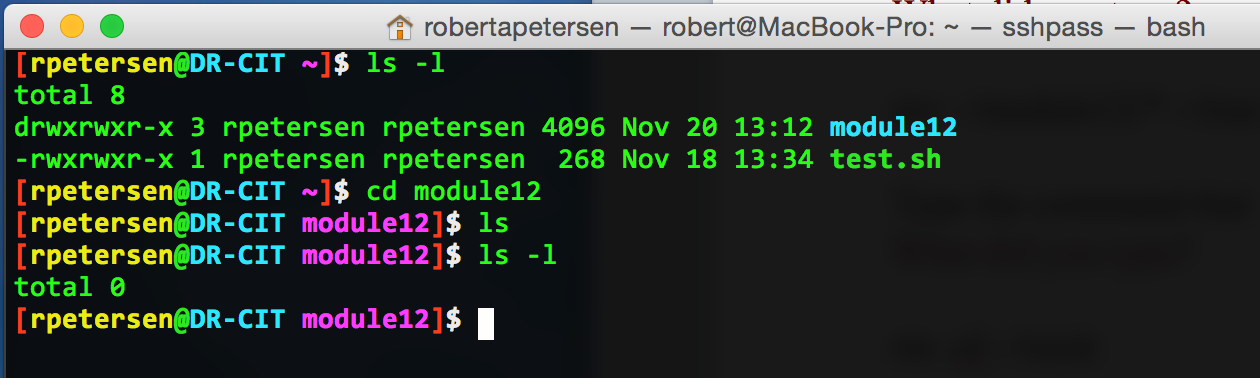
rm -rf ~/trash

Type the command that will echo onto the screen: All files are deleted!

What did you type?

echo ‘All Files Are Deleted!’

Check to see if your file is gone and so is the new directory trash

Insert screenshot here

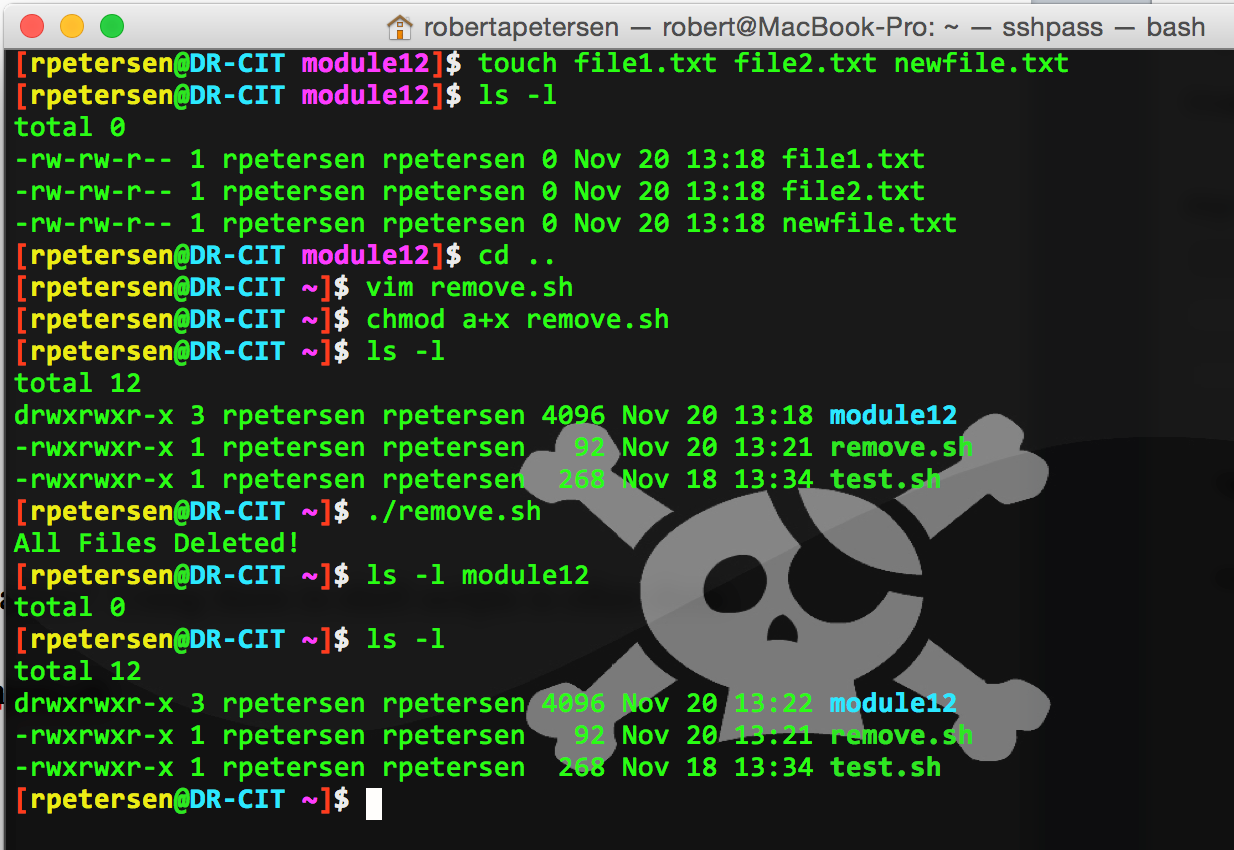
Okay let's put this to use. But first make files that you can remove.

Type: *touch file1.txt file2.txt newfile.txt*

Be sure you have three files there by using ls.

Use vim to create the script above called **remove.sh** that will:

Make a directory called trash, put all your files into the directory trash and then will remove the directory and all its content. Afterwards it will tell you all files are deleted. Be sure to use #! /bin/bash in the script and use chmod to make it executable.

Insert screenshot here of your script after you are sure it works.

Let's have some more fun!

Remember last week when we talked about variables? Using them in shell scripts is often done.

3. Use vim to create the following file called **whatis.sh**

#!/bin/bash  
x=12  
echo "The value of variable x is $x"

Use chmod to make it executable and run it.

What you have done here, is to give x the value of 12. The line echo "The value of variable x is $x" prints the current value of x. When you define a variable, it must not have any whitespace in between the assignment operator: "=".

So the script creates a variable. Is it still good after the script runs? Let's see if it is.

Type: *echo $x*

Is 12 still assigned to x?

No.

4. Type the following script called **whatis2.sh**:

#!/bin/bash

x=12 # assign the value 12 to variable x

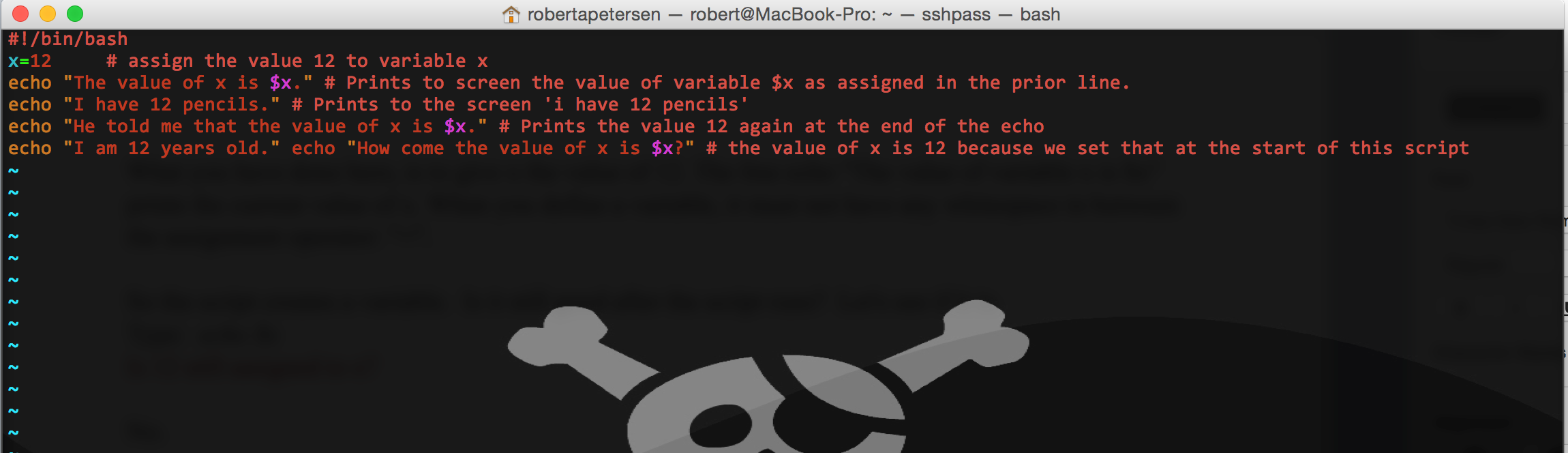
echo "The value of x is $x."

echo "I have 12 pencils."

echo "He told me that the value of x is $x."

echo "I am 12 years old." echo "How come the value of x is $x?"

Chmod the script and run it.

Use vim to add comments to the rest of the lines (I did the first one) explaining what each line is doing. Insert screenshot here.

Let's do a script that let's you change that value of x

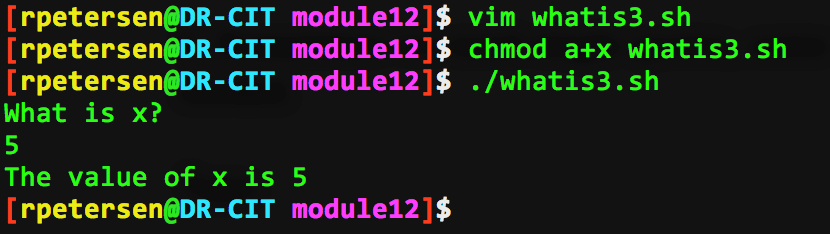
5. Type a script called **whatis3.sh**

#!/bin/bash

echo "What is x?"

read x

echo "The value of x is $x"

Chmod the script and run it, assigning the variable to x as anything you want. Insert a screenshot of your results here.

6. Sometimes we need single quotes, sometimes double quotes and sometimes no quotes are really needed.

Type:

*myname='cathy house'*

*echo 'My name is $myname'*

*echo “My name is $myname”*

Double quote: Variables are expanded when enclosed in double quotes

Single quote: Variables within single quotes are not expanded

7. But there is one other option!! But first let's look at the how we would handle arithmetic in a script. At the command prompt type:

*echo 6 + 5*

echo '6 + 5'

Try this command that evaluates expressions (be sure to include white space between the + sign):

*expr 6 + 5*

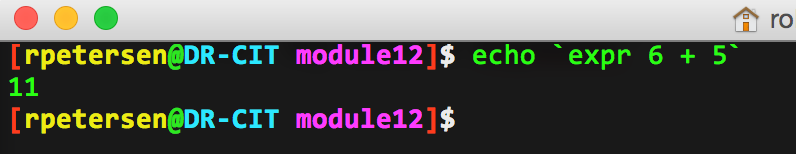
Type (using single quotes and spaces):

*expr 6 + 5*

Did you get 11? Change it to double quotes ”. Did that work?

No, got an output of just 6 + 5

8. Sometimes we need to use something instead of a quote. This is when we want a command to act like a command and not just a word. In this case we need to use the back quote (I call it a tick) You will find the tick under the ~ on your keyboard.

Do the echo command above using the tick and insert a screenshot here.

Type the following:

*expr 10 \\* 3*

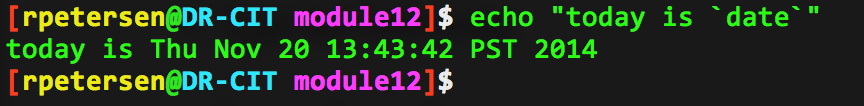
What happens if you remove the \? What does the \ do?

syntax error, the \ lets it know we are using \* as multiplication as opposed to a wildcard.

So we see that when enclosed in single quotes text is treated very literally, but when enclosed in double quotes variables and commands in backward quotes are evaluated and their values output. Above the \ says “Don't take that literally!”

9. Type the following:

*echo "Today is date"*

Using what you know about the tick, how would you make this actually print out the date? Insert a screenshot here.

10. Type the following on the screen:

*echo -e "An apple a day keeps\naway doctor"*

Do the same command and remove the -e. What happens?

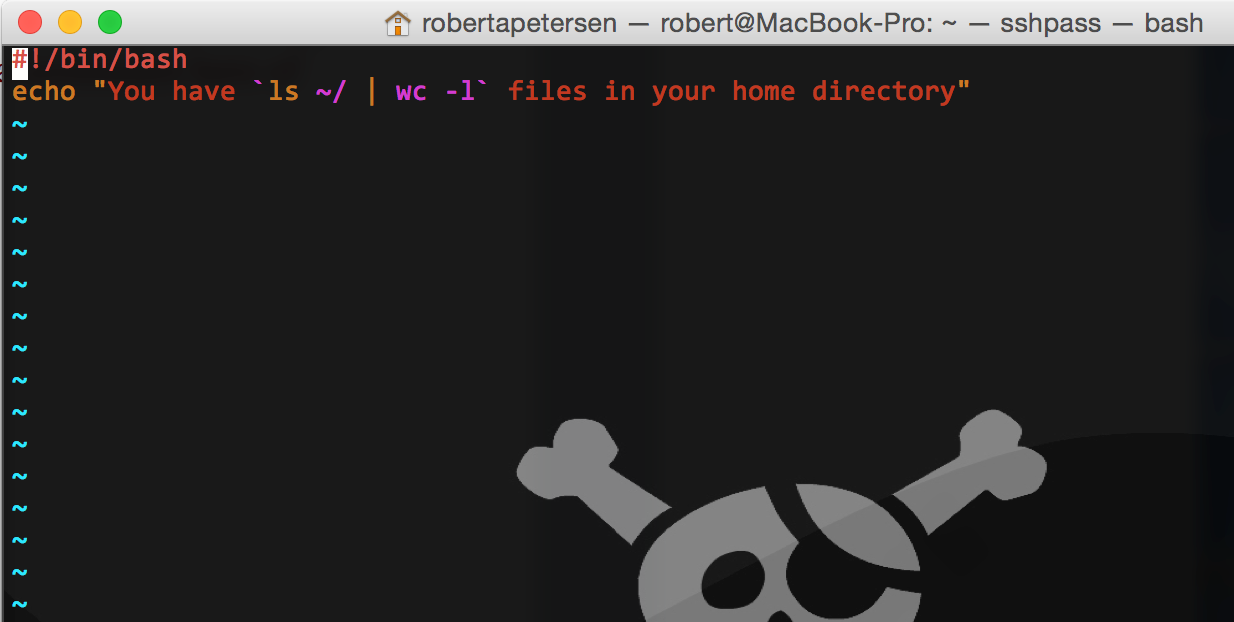
“An apple a day keeps\naway doctor” gets printed, it doesn’t interpret \n as a new line.

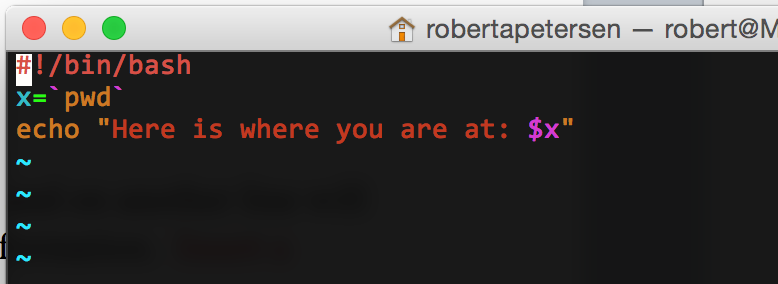
Do the command again with the -e and remove the \n. What happens?

Prints it all to one line.

Use the man pages to tell me what it says about using -e with the echo command.

enables interpretation of backslash escapes

11. Create a script called **list.sh** with the proper permissions that lists the files in your home directory, counts them and echo’s how many files you have in your home directory. Insert a screenshot here of your script.

12.. Create a script with the proper permissions that will put your path (where you are at) into a variable and echo on the screen: Here is where you are at: Again, insert a screenshot of your script here.

13. Write a script called math.sh that will add up 6 + 4 and tell you what it is and on another line will divide 6 by 4 and show you the remainder. Both lines should echo out this information. Insert a screenshot here of the script using cat.