R · I · T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

4002-XXX
Software Development on Linux Systems
Lab 05 – Scripting for Development

Name: _____ **Section:** _____

Activity – Bash Scripting

Linux comes with several scripting languages, the most common of which is Bash. Bash is the Bourne Again Shell, which is an updated version of the Bourne shell.

Basic Bash Review

Most command line commands in Linux are scripts or programs that run. Bash is a scripting language that allows you to combine all of these programs into a runnable script. As most of these programs accept parameters, bash allows the user of variables and input that can be used to send parameters to these programs. This creates a very powerful language you can run operating system commands with.

We will review Bash scripting. You may use either Ubuntu or Fedora for this lab.

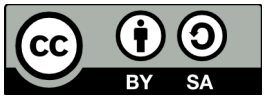
1) Open terminal
2) Type "echo hi" and hit enter [echo is a print statement in Bash]
   This will output "hi" to the screen

3) Type "cat /etc/hosts" and hit enter
   This will output the hosts file which is a list of host configurations for networking

4) Now we will build a script to repeat the above commands
   Open the editor of your choice (recommended gedit or vi [vim])

   For the very first line [must be the very first line] of the file type
   #!/bin/bash

   #! is pronounced sha-bang and tells Linux that it is going to specify a language to run
   /bin/bash is the location of the bash program. You will see other languages do this, such as perl or python.



5) For the second line type
Your last name

means the line is a comment, not to be confused with #! which means a specified language

6) On the next two lines type
echo hi;
cat /etc/hosts;

7) Save the file as “testscript.sh” - It should look like this
#!/bin/bash
last name

echo hi;
cat /etc/hosts;

8) In terminal cd to the directory you saved the file in
type “bash testscript.sh”

This will run the script. The output will probably look something like this, but will vary by machine.

hi
127.0.0.1 localhost

 # The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

9) Now change the permissions of the file to 755
This will make the file executable.

10) Now that the file Is marked executable you make invoke bash with the command ./
In terminal in the the directory the file is in type
"./testscript.sh"

This will now run the file for you and will have the same results as "bash testscript.sh"

Show the results of all of these to your instructor/TA to get credit

Instructor/TA Sign-Off _____

Bash Review 2

We will now review variables and logic. Variables are referenced by name or value in Bash.

1) Variables

I) Open terminal and type "testvariable=100" and hit enter
[bash does NOT accept spaces between the variable name, equals sign and value]

The name of this variable is now "testvariable"
The value of this variable is now "100"

The name of the variable can be referenced by "testvariable"
The value of this variable can be referenced by "$testvariable

II) Now type "echo testvariable" and hit enter
This should display "testvariable"

III) Now type "echo $testvariable" and hit enter
This should display "100"

IV) Using the editor of your choice (preferably gedit or vi [vim]) type the following

```
#!/bin/bash
#Script to check variables that have been exported and variables that have not

testvariable3=”yes”;

echo $testvariable1;
echo $testvariable2;
echo $testvariable3;
```

V) Save this script as testvariable.sh

VI) Open terminal and cd to the directory you saved your script in

VII) Type “testvariable1=no”

VIII) Type “testvariable2=maybe”

IX) Type “export testvariable1”

Export allows a variable to be used by any scripts or processes you run within that terminal session. Outside of this terminal session, the variables will not exist. When you close the terminal session the variables will also no longer exist.

X) Now type “bash testvariable.sh”

The output should display

no

yes

no is displayed because it has been exported for use in script. Yes is displayed because it is a variable that is defined inside of the script. Maybe is not shown because it was declared outside of the script and was not exported.

Similarly, variable exported within a script can then be used by scripts invoked by the original script.

2) Logic statements Part 1

If statements in bash look like normal if statements, but are space sensitive and follow a slightly different syntax than standard programming languages. If statements start with the word if and are ended by the word fi.

Variables need a space between the variable, operator and value in an if statement. There must also be a space before and after the if statement brackets. If you if statement fails it is probably due to incorrect spacing, incorrect operators or missing dollar signs/typos.

Note: String variables must have quotes around them in the if statement while numerical variables must not.
Example of string variable: if ["$myString" == "yes"]
Example of numerical variable: if [$myNum -eq 10]

Reference table for operators borrowed from http://tldp.org/LDP/abs/html/refcards.html

Arithmetic Comparison			String Comparison
-eq	Equal to	=	Equal to
-ne	Not equal to	!=	Not equal to
-lt	Less than	\<	Less than (ASCII) *
-le	Less than or equal to		
-gt	Greater than	\>	Greater than (ASCII) *
-ge	Greater than or equal to		
		-z	String is empty
		-n	String is not empty

I) Create a new script in the same folder as your testvariable.sh script. Call this file testlogic.sh

II) Inside of your script write the following

```
#!/bin/bash
#Script to test if statements

myNum1=10;

if  [ $myNum1 -eq 10 ]
then
      echo $myNum1;
else
      echo “Num variable did not equal 10”;
fi
```

III) Save the file as testlogic.sh

IV) In terminal cd to the directory of your files and type “bash testlogic.sh”
The output should display
10

V) Now modify testlogic.sh to include elif (else if). It should look like this

```
#!/bin/bash
#Script to test if statements

myNum1=10;

if  [ $myNum1 -eq 10000 ]
then
      echo “Number is 1000”;
elif  [  $myNum1 -gt 0  ]
then
      echo $myNum1;
else
      echo “Num variable did not equal 10”;
fi
```

VI) Save the script and run it

The output should display

10

VII) In Bash, command line arguments are automatically taken is as number

${1} is the first argument, ${2} is the second argument and so on

Curly braces { } mean to expand the values in a location. This is used because the arguments are an array and you are calling a position.

Modify your script to print command line arguments. It should look like this.

```
#!/bin/bash
#Script to test if statements

if  [ "${1}" = "one" ]
then
    echo "Number is ${1}";
elif  [  -z ${1}  ]
then
    echo "Number is empty";
else
    echo "Number was not one and was not empty";
 fi
```

VIII) Save the file and run it as "bash testlogic.sh one"

The output should display

Number is one

IX) Run it again as "bash testlogic.sh"

The output should display

Number is empty

X) Run it again as "bash testlogic.sh two"

The output should display

Number was not one and was not empty

Show the results of all of these to your instructor/TA to get credit

Instructor/TA Sign-Off _____

R · I · T

Rochester Institute of Technology
Golisano College of Computing and Information Sciences
Department of Information Sciences and Technologies

3) Logic statements Part 2

I) In the same folder as your testvariable.sh script, create a second script called echovars.sh . The contents of this script should look like this

```
#!/bin/bash
#Script to print variables from a subprocess

echo $testvariable2;
echo $testvariable3;
echo “end of script”;
```

II) Now modify your original testvariable.sh script to look like the following

```
#!/bin/bash
#Script to check variables that have been exported and variables that have not

 testvariable3=”yes”;

 if [ "$testvariable2" = "" ]  #there are spaces between the variable name, equals
 then                          #sign and quotes in if statements
     testvariable2="maybe";
     export testvariable2;
      echo “exported new variable 2”;
      bash echovars.sh;
 else
     echo "test variable 2 was empty. Script did not run";
 fi
```

III) Save the script

VI) cd to the directory of the scripts and run type “bash testvariable.sh”

The output should display

Exported new variable 2

maybe

end of script

Maybe was displayed because variable 2 was exported. Variable 3 was not displayed because it was not exported from the testvariable.sh script.

R · I · T

Rochester Institute of Technology
Golisano College of Computing and Information Sciences
Department of Information Sciences and Technologies

4) Logic statements Part 3

Loops in Bash are similar to normal loops, but follow the same coding syntax as Bash if statements.

1) In the folder with all of your scripts create a new file called looptest.sh

2) Make that file look like this

```
#!/bin/bash
#Script to show looping. It will take in the results of the command ls as a variable and
#iterate through it.

for item in $( ls );  #item is the variable name given to each item in the loop
do
      echo $item;
done
```

3) Save the file and run it

Your output should display the same results as ls -1 (that's a one this time, not an L)

4) Now modify your script for a while loop. You will use the let command. The command let allows you to perform arithmetic on variables that you would not normally be able to do in bash.

It should look like this

```
#!/bin/bash
#Script to show looping. It will take count to 10 using a while loop

counter=0;
while [ $counter -lt 5 ];
do
      echo $counter;
      let counter=counter+1;        #the let command allows you to do arithmetic on
                                    #variables
done

until [ $counter -eq 0 ];
do
      echo $counter;
      let counter=counter-1;
done
```

5) Save your script and run it
The output should display
0
1
2
3
4
5
4
3
2
1

6) Now modify the script to make use of arrays. Arrays in bash are similar to normal variables and arrays. Arrays can automatically be made into associative arrays with the use of the -A flag. By default they will be numerically indexed.
in a variable basically means length of in Bash.

The script should look like this

```
#!/bin/bash
#Script to show looping. It will take count to 10 using a while loop

array1=("one" "two" "three" "four" "five");

declare -A array2;
array2["juice"]="grape";
array2["milk"]="soy";
array2["wine"]="white";

echo "There are ${#array2[@]} items in array2";

for i in "${array1[@]}"              #@ sign meets “at all parameters”
do
        echo $i;
done

for item in "${!array2[@]}"          #! before an array means to get the array keys,
do                                   #not the values
        echo ${array2[$item]} $item;
done
```

7) Save and run the script
The output should display
There are 3 items in array2
one
two
three
four
five
white wine
grape juice
soy milk

8) Practice
For the lab write up create a script to take each item from the command ls (hint: **for item in $(ls "/tmp");**) and store it in an associative array as the **key** for that array. Make each **value** for that spot in the array the next number. The first file would be one, the second two, etc (ex: $files["test.sh"]=2). The key and the values will have to be variables.

Then iterate through the associative array and if the **value** is less than 3 print the file name (the array key), else print the value.

Complete this script and hand it in as part of the lab write up.

R · I · T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

Bash Scripting for Development

Bash is an extensive language and has many commands.
For a short list of these commands reference
http://www.gnu.org/s/bash/manual/bash.html#Shell-Builtin-Commands
http://ss64.com/bash/

We will look at some of the commands necessary for this course and building packages. You may use the above references or the man command to look up how to use these commands. For this section of the lab you will use the commands listed below, as well as the commands from the previous section of this lab and previous labs.

Commands:
adduser
chgrp
chown
fakeroot
ln
passwd
tar
uname
whoami

1) adduser

The adduser command does just that, adds a user.
The syntax is “adduser -d their/home/directory -G groups name”
-d is the user's home directory
-G is the groups they belong to. If the group does not exist; it will make one for you.

Ubuntu example “sudo adduser -d /home/jameson -G developers jameson”

Fedora example “su -”
“adduser -d /home/jameson -G developers jameson”

Create a user on your system with the following attributes
Name: **djsmith** and group: **themixers**

Create a second user on the system with the following attributes
Name: **djjack** and group: **themixers**

List your commands in the write up to get credit for this part of the lab

2) passwd

The passwd command changes the password for a user.

Change the password of the user djsmith
In terminal type “passwd djsmith” to change the password for the user djsmith

Change the password of the user djjack as well

List your commands in the write up to get credit for this part of the lab

3) chgrp

chgrp stands for change group and is used to change the group ownership of a file.
The syntax is “chgrp group file1 file2 file3 file4....”
The group is the user group a file belongs to

Ubuntu example “sudo chgrp developers file1.txt”

Fedora example “su -”
“chgrp developers file1.txt”

Create a file in your home directory called “groupchange.txt”

Change the group on that file to **themixers**

List your commands in the write up to get credit for this part of the lab

4) chown

chown stands for change ownership and is used to change the owner of a file.
The syntax is “chown owner:group filename”
The owner is the user who owns the file and the group is the user group the file belongs to.
In many cases the owner and group will be the same because the user will have their own private file group.

This is an example that changes a file's ownership to root and the group to the admin users
Ubuntu example “sudo chown root:admin file1.txt”

Fedora example “su -”
“chown root:admin file1.txt”

Create a file in your home directory called “ownerchange.txt”

Create a file in your home directory called “ownerchg.txt”

Use ls -l (that's minus L) to see the current owner and group for the file

Change the owner of the ownerchange.txt file to djsmith and the group to themixers

Change the owner of the ownerchg.txt file to root and the group to root

List your commands in the write up to get credit for this part of the lab

5) whoami

whoami is a command that lets you know the current user. This can be used by scripts to determine if the user is root or another expected user, such as Oracle.

In terminal type “whoami” to display the current user

Write a bash script to determine who the user is and make an if statement to check if the user is root or not. If the users is not root print “not root”. If the user is root print “root”.
hint: back ticks can be used to run a command as a variable. For example

```
if  [  “`ls`” == “file1.txt”  ]
```

You will need to run the script as a normal user then as a root user. This means running the command as sudo on Ubuntu or running “su -” before running the command on Fedora.

6)fakeroot

fakeroot is a command that allows you to execute commands as if you were a root user, but will not actually change anything. Fakeroot can be used to install files that are designed for root under non-root users. This is commonly used for files that you might worry would have security issues running as root.

As a non-root user in terminal type “whoami” and hit enter
This should show your username

Then as the same user in terminal type “fakeroot whoami” and hit enter
This should show root

Modify your script from the “whoami” section to determine if the user is root. If the user is root.

Turn this script in with the write up to get credit for this part of the lab

7) ln

The ln command stands for link. This is used to create links to files. Links redirect to a file. There are two types of links in Linux. There are hard links and soft links. Hard links are links to the physical hard drive location of a file. If you delete the original file and hard links still exist, the file will NOT be removed from the hard drive. The file will not be deleted from the hard drive until all hard links to that file are removed. Soft links are only links to another file on the filesystem. If you delete the original file, the soft links will no longer work. Soft links are just the path to a file. When that file is deleted, the soft link will become a broken file.

Links are used to provide a different location for a file, where the original is too difficult to move. For example, you may have a server application that only has access to one directory for security reasons, but you may need to give that server access to a log file or database file outside of its directory. This could be solved by putting a symbolic link to the database file within the server's working directory.

Links are also used to provide structure for an application or api. For example, all of your code may be scattered throughout various folders by logical category, such as database access files, interface files, application managers, etc. To make it easy to interface with these files, you could make one directory that has symbolic links to all the files you need to work with. This allows you to call any of the files you need from one directory, regardless of their location.

Syntax for hard link: “ln /full/path/to/file /full/path/to/new/link”
Example “ln /etc/hosts /home/user/hosts”

Syntax for soft link: “ln -s /full/path/to/file /full/path/to/new/link”
Example “ln -s /etc/hosts /home/user/hosts”

In your home folder create a file named “link1.txt” and a file named “link2.txt”

Create a hard link in the same directory to link1.txt called “linktofile1.txt”
(You may use the command “pwd” to get the full path)

Create a soft link in the same directory to link2.txt called “linktofile2.txt”

In your home directory type “ls -l” to get a listing of the files

remove link1.txt and link2.txt

Then type “ls -l”

Explain what happened to linktofile1 and linktofile2 as part of the lab write up

8) uname

The uname command lists all of the information about the operating system. This includes the OS type, kernel version and other data.

The uname command is important because it tells you whether or not the system is 32 bit or 64 bit, and which kernel is being run. These are important to know for an installer.

In terminal type “uname -a”
The a means all fields and will give you all the information.

To determine if the operating system is 32 bit or 64 bit type “uname -m”
The m means machine type.
i686 is 32 bit
i386 is older 32 bit
x86_64 is 64 bit

To determine the kernel version of the OS type “uname – r”
The r stands for release version

Write a bash script to determine if the kernel version is 32 bit or 64 bit
hint: remember you can use back ticks to run a command as a variable
Example: “if [**“`ls`**” == “file1.txt”]

9) Tar

The tar command is a compression command similar to zip. Tar allows you to multiply compression by combining the tar compression with gunzip compression; a zip algorithm.

Compressed tar files are called tarballs. They usually end in tar.gz
The tar stands for the tar compression

You may create a tar file with
"tar -cvzf compressedfile.tar.gz folder" (you may list filenames instead of a folder)
c stands for create
v stands for verbose (tells you when the file is compressed, instead of no feedback)
z means zip with gzip for multiplied compression
f stands for files (files or folders to compress are listed after the f)

You may untar a file with
"tar -xvzf compressedfile.tar.gz"
x stands for extract

In the folder where you made all of your scripts create a tar file of only your scripts
Name the tar file **yourlastname**.tar.gz where yourlastname is your last name.
(hint: for the files to be compressed use *.sh since all of your scripts end in sh)

Note: You will need to know this command for the packaging installers later in the course

Hand in this tarball with all your scripts as part of the lab write up. If all of your scripts are in this tarball, this will count for handing in all of the scripts in the previous activities.

10) Activity **Turn this script in with the rest of the lab write up to receive credit**

Note: The commands in this script will use many of the commands you used in lab 3

I) In your home directory create a folder called “packagetest”
II) cd to the new “packagetest” folder
III) Create a folder called “release_64”
IV) Create another folder called “release_32”
V) Create files named “r32_1.bin” and “r32_2.bin”
VI) Create files named “r64_1.bin” , “r64_2.bin” and “r64_3.bin”
VII) Tar the five bin files into a tar called “release.tar.gz”
VIII) Write a script called install.sh that
a) accepts 3 arguments as strings (IE: “bash install.sh “word1” “word2” “word3”) [this automatically happens as ${1} ${2} and ${3}]
b) determines if the user is root or not
c) if the user is root, then start a script called “release.sh” that passes the parameters to it as arguments.
d) if the user is not root, then start the script “release.sh” with fakeroot and pass the parameters to it as arguments.

IX) Write a script called release.sh that
a) accepts the three arguments from the install.sh script (automatically happens)
b) has a variable called done that is the string “finished”
c) stores the parameters in an array called files (you can store them as files[0], files[1] and files[2])

d) determines if the machine is 32 bit or 64bit
e) -If the machine is 32bit, for each argument create a file in with the name that was passed in as an argument in the “release_32” folder.
-Untar the file release.tar.gz and move the files into the “release_32” folder
-Changes ownership of the “r32_1.bin” file to user djsmith and group themixers
-Changes the group of the “r32_2.bin” file to themixers
-Creates a symbolic link of “r32_2.bin” in the “packagetest” folder called “32bitrelease”

f)-If the machine is 64bit, untar the file release.tar.gz and move the files into the folder “release_64” folder.
-Changes the ownership of the “r64_1.bin” file to root and the group to root
-Changes the ownership of the “r64_2.bin” file to user djjack and group themixers
-Changes the group of the “r64_3.bin” file to group themixers
-Creates a hard link of “r64_3.bin” in the “packagetest” folder called “64bitrelease”

g) Prints the variable you made called $done