# Overview

## Goals

Scripts are a powerful way to automate repetitive or complex tasks. They allow for more efficient and error free implantation of new services and configurations.

Note that # or $ will be used as short prompt on the terminal

Be sure all the scripts have the proper header. The ones listed on the example scripts are generic (e.g bourne). Make sure the top two lines for each script are the proper she-bang and comment line for your script. The comment line(s) must be at least your name, the lab section and a brief description of the script.

## Deliverables

* Lab Report with the following
  + Documented results of running each script
    - Show script working
    - Show script rejecting errors
  + Include copy of the script text
    - Screen captures are not allowed for text items

# Using Shells and Scripts

## Getting started

1. Start the Debian VM
2. Verify your VM does use bash as the default for your userid
3. Create a “header” file containing the she-bang for bash
   1. which bash > bashheader.sh
   2. Edit bashheader.sh and start the line with #! (no space)
   3. Add a comment with your name, class name and lab section
      1. Show a copy of this template in your report\*\*
   4. Use this file as a template for the first 2 lines for all your bash scripts

***Note:*** The she-bang in the following scripts is GENERIC, you must substitute the correct data for your scripts to work!

***Note:*** The name of the script is meaningless to the shell. “.sh” is usually used to tell a human this is an executeable script.

## First Script

#!/bin/sh

# your comment

echo "Today is `date`"

echo"I am user $USER"

echo"I am using the $TERM terminal"

The first script will report today’s date, who is logged on, and the terminal type being used:

The name of the script should be *myfirst.sh*. Use the header created as a template to start the script. Be sure to replace the first two lines with the proper data from your template header information. Be sure to do this for all the following scripts also.

To run the script, first make it executable, and then run it:

$chmod +x myfirst.sh

$./myfirst.sh

Note: if your PATH has the directory that the script is in then the “./” is not needed before the file name to execute it. The full path name may also be used to run the script. Don’t forget to make all scripts executable before running them.

In the report show your complete script listing along with the result of running the script.

## Reading Script

This script will read from the keyboard and do an action based on the input.

#!/bin/sh

# some comment

echo "Host:"

read hostn

echo "User ID:"

read uid

echo "File name:"

read filen

command="scp $uid@$hostn:$filen $filen"

echo "executing $command:"

$command

Name the script *myscp.sh*. Make it executable and test it.

In your report show the complete script and the result of running it. Describe briefly what the script does.

As you type in scripts you will have typos and encounter various errors. After this script is running induce an error by putting blanks around the equal sign. Note the error you get.

## Command line arguments

This script will accept three arguments and use them to complete a command, chmod in this case.

#!/bin/sh

# this script will add the permissions for a file

# the first parameter is for who: u, g, o, or a

# the second command is for the permission: r, w, or x

# the third parameter is the file name

command="chmod $1+$2 $3"

echo adding permissions with the following command: $command

$command

Name script *addperm.sh* and test it. Document the results.

## Using if, test and exit

Enhance *addperm.sh* by adding a simple check to verify the right number of parms have been passed and exiting if not.

#!/bin/sh

# this script will add the permissions for a file

# the first parameter is for whom: u, g, o, or a

# the second command is for the permission: r, w, or x

# the third parameter is the file name

if test $# –ne 3

then

echo Need 3 parameters

exit 1

fi

command="chmod $1+$2 $3"

echo adding permissions with the following command: $command

$command

Copy *addperm.sh* to *addpermplus.sh*, and then add the code for the check.

Addpermplus.sh is not easy to type, create an alias for it:

alias app="~/addpermplus.sh"

Test the alias *app*. Then try changing to another directory and try the command *app* again. Note the ~ in the alias, what does this imply? Think of how you run a file that is not in the pwd. Document the alias working.

## Case

The next script uses the case statement to check if a character is upper or lower case, and if it is numeric or otherwise.

#!/bin/sh

# some comment

echo; echo "Hit a key, then hit return."

read keypress

case "$keypress" in

[[:lower:]] ) echo "Lowercase letter";;

[[:upper:]] ) echo "Uppercase letter";;

[0-9] ) echo "Digit";;

\* ) echo "Punctuation, whitespace, or other";;

esac

Name the script *case.sh.* Test and document the code working.

Make one enhancement to note if the character is one of the common punction characters (.,?!;:’). Document and show the enhancement working.

## Using expr to check string length

This script will check the length of an argument string. Name this script *strlench.sh*.

#!/bin/sh

# some comment

slen=`expr "$1" : '.\*'`

echo $1 $slen

if [ $slen -lt 4 ] ; then

echo "String too short"

elif [ $slen -gt 10 ] ; then

echo "String too long"

else

echo "Goldilocks"

fi

Test the script, then modify it to test 2 arguments, the first is to be 2 to 8 chars long, the second 5 to 8 chars long. Document and show the modified script running.

Create another error by removing the backticks from the expr and see what error you get. Document your results.

## Using for/while

This script will do two simple loops. First it will backup from a list of files, and then wait for a file to become readable. Name this script *backWait.sh*

#!/bin/sh

# some comment

for file in `cat filelist`

do

cp $file $file.bak

done

while true ; do

[ -r $1 ] && break

sleep 1

done

Create a file named *filelist* with four filenames in it: *file1*, *file2*, *file3*, and *file4*. The names only need be separated by whitespace. Quickly create *file1* and *file3* by using *touch* (touch file1). *Touch* updates a file’s current timestamp. If the file does not exist *touch* creates a zero length file with the current timestamp. Create another file that has no read attributes. For example the file name could be *noread*. To change that file so no one has read ability use the command: chmod -r noread. Run the *backWait.sh* with the file name for the no read file as a parameter: backWait.sh noread. The script should then attempt to backup the listed files in *filelist*. Note that it will fail for those that do not exist. The script will then wait for *noread* to have read attributes. Open a second terminal and change *noread* to read. The script should then complete. Document the backup worked.

## Trapping

This next script will be a little "dangerous". It will intercept the **interrupt** and **stop** signals (<ctrl>-c and <ctrl>-z) and echo a simple message. You may name the script as you wish, but document the name.

#!/bin/sh

# some comment

trap "echo I am done" SIGINT SIGSTOP 20 24

echo "pid is $$"

while :

do

sleep 5

done

In its current state the program cannot be stopped with the normal <ctrl-c> or even <ctrl-z>. Note the <ctrl-z> works a little differently than <ctrl-c>. Since the interrupts are now replaced with an echo you will need another way to stop the program. You will need another terminal with root authority to kill the program. When the program is run note the pid reported. On the other terminal get root authority and issue the following command:

sudo kill nnnn

Where nnnn is pid of the scripts. Note the above user had sudo enabled. If you don't have sudo enabled use su – , then issue the kill command. After the kill command is executed the script should stop and terminal control regained.

## Functions

#!/bin/sh

# some comment

function quit() {

echo Bye!

exit

}

function e1 {

echo $1

}

e1 Dog

e1 Cat

quit

echo here?

The script to the right is a simple script (*func1.sh*) with two functions: quit and e1. Quit has no arguments, e1 expects one argument.

Note that when the script is executed the quit stops the script before the last echo statement.

Modify the script so function e1 uses two arguments in its processing. Make the use interesting. One possibility is to add two numbers.

## 

## Shell 1Deliverables:

1. Printed report with all script listings and results
   1. Show results of running scripts
      1. Show good data being used
      2. Show proper rejection of bad data (as appropriate)
   2. Be sure to answer questions and document requested changes.
2. Moodle
   1. Upload copy of lab report

# Preview Scripts II: Writing Your Own Scripts

Here is a preview of the Scripts II lab (later in the semester). As you do the Scripts I lab think of which of those scripts you can modify and use to do Scripts II.

## Improve addperm.sh

Copy addperm.sh to create a new file, addpermplus.sh.

Add checks to addpermplus.sh

* the first argument value is only u, g, o or a
* the second argument value is only r, w, or x
* the third argument is a valid name
  + What do *you* need to check for?
  + Justify your choice.

Be sure to issue meaningful errors.

Test your script and document the results.

## Create new accounts

Your script will add new users from a file. The origin of the data will be an Excel (or equivalent) spreadsheet. The spreadsheed data can be easily be exported as a specially formatted text file called a CSV (comma separated variables) file. Normal capitalization will be used. Your delimiter is the same as one produced by making a CSV file from an Excel spreadsheet (or equivalent). Your script will read the exported csv file, change all upper case letters to lower case and form a userid with the following algorithm: the userid is to use up to the first 6 chars of the last name concatenated with the first 2 chars of the first name. You can use the **adduser** or **useradd** functions to create the accounts. Generate a random password that can be recorded and sent to the user. Use any method you want to generate the random pw. Make sure the user has a home directory the same as their userid.

Document your code and the successful testing of it. Use a real spreadsheet to hold the users name in the following format:

|  |  |
| --- | --- |
| fname | lname |
| Tony | Kombol |
| Fred | Ziffle |
| Darren | Kitchen |
| Shannon | Morse |

Create your own table with your own names in them. Have at least 10 entries. One way to document the generated id and the pw would be to create a csv file with the users name, id and pw. Then it could be easily read as text or with a spreadsheet.

You can use a combination of sed, bash script, awk, whatever to automate the task.

# Scripts II Deliverables:

1. Document the above work in a printed report
   1. Listing of all scripts
   2. Listing showing scripts working (including bad input failing)
2. Moodle
   1. Upload copies of your scripts to Moodle
   2. Upload a copy of your report