UNIX Systems

Homework 1 – Bash Basics

Abraham Murciano

March 12, 2021

Question 6. If a user wants to execute a `script.sh` without a shebang line or execute permissions, what should the user type?

Answer. This will invoke the bash interpreter with `script.sh` as a parameter. Since the bash interpreter has execute permissions, it is able to run, read the file passed to it as a parameter, and run the commands in it treating them as bash commands, since after all, it *is* the bash interpreter.

```
$ bash script.sh
```

Question 7. Which command is being run in this script to check if `file.txt` exists?

```
if [ -f file.txt ]; then
    echo "file.txt exists"
fi
```

Answer. It is the `-f` which checks if the file exists. This also checks that it is a regular file. I.e. it is not a directory or device.

Question 8. What happens if you use `set -e` in a Bash script?

Answer. As this command shows, the `-e` flag of the `set` command sets the shell option to immediately exit the execution of the script if any command returns a non-zero value, indicating that it failed.

```
$ set --help | grep "\-e"
-e  Exit immediately if a command exits with a non-zero status.
errexit     same as -e
```

Question 9. How does the `SUID` or `setuid` bit affect executable commands?

Answer. When the command is executed, its running privileges elevate to the user owner of the command.

Question 10. To keep a loop going until a certain condition becomes true, what would you likely use?

Answer. A while loop would be appropriate for this scenario. The condition of the while loop would be the negation of the condition you want the loop to run until.

Question 11. The `data.txt` file is owned by `root:root` with `rw-------` permissions. The script will be executed by `user1`. Are the two conditionals in this script the same? Explain.

```
[[ -e data.txt ]] && cat data.txt || echo "data.txt doesn't exist"
if [[ -e data.txt ]]; then
    cat data.txt
else
    echo "data.txt doesn't exist"
fi
```

Answer. The two conditions are different. In the first one, the test `[[ -e data.txt ]]` succeeds, since the file exists, then proceeds onto the `cat` command. This command then fails since `user1` does not have permission to read the file. Since the left hand side of the `||` command returned false, the right hand side runs.

However, in the `if` statement, the condition succeeds so it continues to the `cat` command. This fails for the same reason as above, but control never reaches the else, since the condition was true.

Question 12. In order to write a script that iterates through the files in a directory, which of the following could you use?

Answer. The correct loop is the following one.

```
for i in $(ls); do
    # ...
done
```

Question 13. What is the difference between these two conditional expressions?

```
[[ $A == $B ]]
[[ $A -eq $B ]]
```

Answer. `[[ $A == $B ]]` is used for text comparisons whereas `[[ $A -eq $B ]]` is used for numeric comparisons.

Question 14. What is the output of this command sequence? Explain.

```
cat <<EOF
--------------------------
This is line 1.
This is line 2.
This is line 3.
--------------------------
EOF
```

Answer. The output of this command sequence is displayed below. The reason for it is that the `cat` command is first given an end of file, then it is given a few more lines followed by another end of file. The `cat` command concatenates the empty 'file' with the next few lines and prints that result...

Question 15. Given the following,

```
$ ll
total 0
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file1.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file2.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file3.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file4.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file5.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file6.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  file7.text
```

What is the output of this line?

```
$ ll | sed -e 's,file,text,g'
```

Answer. This `sed` command takes as input the output of `ll` as shown above, and replaces all instances of the word 'file' with the word 'text', producing the following output.

```
$ ll
total 0
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text1.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text2.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text3.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text4.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text5.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text6.text
-rw-r--r-- 1  danzig staff  0  Jul 12 19:30  text7.text
```

Question 16. What is the output of this script?

```
#!/bin/bash
fname=john
```

```
john=thomas
echo ${!fname}
```

Answer. This is an 'indirect expansion'. If the first character of a parameter is an exclamation mark, then a variable whose name is the value of the following variable is used instead. In this example, the variable `john` is echoed.

Question 17. What is the difference between the `$@` and `$*` variables in Bash?

Answer. `$@` treats each quoted argument as a separate entity. `$*` treats the entire argument string as one entity.

Question 18. Is there anything wrong with the following script?

```
#!/bin/bash
read -p "Enter your pet type." PET
if [ $PET = dog ]; then
    echo "You have a dog"
fi
```

Answer. There is nothing wrong with it. The condition checks the value of `PET` perfectly.

Question 19. What does the following script accomplish?

```
#!/bin/bash
declare -A ARRAY=([user1]=bob [user2]=ted [user3]=sally)
KEYS=(${!ARRAY[@]})
for (( i=0; $i < ${#ARRAY[@]}; i+=1 )); do
    echo ${KEYS[$i]} - ${ARRAY[${KEYS[$i]}]}
done
```

Answer. It creates an indexed array of the associative array named ARRAY. It then uses a C-style for loop and the indexed array to loop through all items in the associative array, outputting the key and value of each array item using the index number.

Question 20. How can you gather history together for multiple terminals?

Answer. When one closes a terminal session, the commands from that session are appended to the history file, overwriting any previous appends that were performed since that terminal session was opened...

Assuming none of the terminal sessions were running simultaneously, the `history` command will show all the history across all sessions.

Question 21. Given the listed permissions on `data.txt`, is it possible that `user2` could have read, write, and execute permissions, on `data.txt`?

```
$ ls -rlt
total 1
-rw-rw-r--+ 1 user1 user1 0 Oct 20 09:23 data.txt
```

Answer. Yes, the `+` at the end of the 10-digit permission string signifies there's an access control list. This could possibly give `user2` permissions not visible by `ls -l`.

Question 22. What will be the output of this command?

```
$ ls -1
backup 2018-Apr.tar
backup-- 2018. Apr. tar
backup 2018-Apr.tar
backupa2018-Apr.tar
backup-A2018-Apr, tar
$ ls -1 backup[^[:lower:] [:digit:]][[:punct:]][[:alpha:]]*.tar
```

Answer. This will produce an error saying that the file does not exist, since the regular expression does not match any file.

Question 23. In Bash, what does a `#` at the end of the default prompt string indicate?

Answer. The `#` usually indicates that the user is acting as the root user.

Question 24. Which file allows you to save modifications to the shell environment across sessions?

Answer. Most shells use the `~/.profile` file, however Bash uses a file called `~/.bash_profile`

Question 25. When used from within a Bash script, which variable would contain the name of the script?

Answer. `$0` refers to the first parameter given to the shell prompt when running a script. The first parameter is always the name of the command or script to run.