Linux shellscripting labs

Table of Contents

Exercise Module 1. Introduction.

Level 1.

Ask your instructor for a login account and a password and log in.

Ask your instructor for a login account and a password and log in.

If you are in a graphical environment, open a terminal and check your current working directory

1. Create a subdirectory called 'scriptdir' in your login dir.

2. Check your shell.
   If you are not in bash, then start a shell.

3. Create a script called 'list.bash' in scriptdir that runs in the bash shell and lists all files.

4. Run the script.

5. What is the return code after the script has finished?

6. Create a script called retcode.bash in scriptdir, that only sets the return code to 5. Make the script executable and run it. Check the return code twice in a row. It should be '5', the first time, and '0' the second time.

7. Create a script called proclist.bash in scriptdir, that runs the 'ps -f' command. When you execute the script, what do you see?

8. Run the following command

```
$ exec sleep 3
```

What happens?

Level 2.

Ask your instructor for a login account and a password and log in.

Ask your instructor for a login account and a password and log in.

If you are in a graphical environment, open a terminal and check your current working directory

$ **pwd**

1. Create a subdirectory called 'scriptdir' in your login dir.

```
$ mkdir ~/scriptdir
```

2. Check your shell.

```
$ ps
PID TTY  TIME CMD
2298 pts/4   0:00 ps
1669 pts/4   0:00 bash
```

If you are not in bash, then start a shell.

$ **bash**

3. Create a script called 'list.bash' in scriptdir that runs in the bash shell and lists all files.

```
$ cd scriptdir
$ vi list.bash
#!/bin/bash
ls -lai
:wq

$ chmod +x list.bash
```

4. Run the script.

```
$ ./list.bash
351569 drwxr-xr-x    2 root   root 512 Oct 12 10:52 .
368860 drwxr-xr-x    3 root   root 512 Oct 12 10:51 ..
351585 -rwxr-xr-x    1 root   root 23  Oct 12 10:51 list.bash
```

5. What is the return code after the script has finished?

```
$ echo $?
0
```

6. Create a script called retcode.bash in scriptdir, that only sets the return code to 5. Make the script executable and run it. Check the return code twice in a row. It should be '5', the first time, and '0' the second time.

```
$ vi retcode.bash
#!/bin/bash
exit 5
:wq!

$ chmod +x retcode.bash
$ ./retcode.bash
$ echo $?
5
```

7. Create a script called proclist.bash in scriptdir, that runs the 'ps -f' command. When you execute the script, what do you see?

```
$ vi proclist.bash
#!/bin/bash
ps -f
:wq!

$ chmod +x proclist.bash
$ ./proclist.bash
UID  PID  PPID C  STIME    TTY          TIME CMD
root 2335 2334 0  11:01:03 pts/4      0:00 ps -f
(output skipped)
```

8. Run the following command

```
$ exec sleep 3
```

What happens?

Answer: The code of the shell will be replaced by the sleep command. After the sleep command finishes, the process will exit.

Time Frame:
About one hour is needed for this lab.

Prerequisites:

Objectives:

Exercise Module 3. Redirection & command-line piping

Level 1.

1. Create an empty file using output redirection.

2. Use two different methods to put the following text into a new file, without using an editor: 'straight from keyboard into file'.

3. Again without using an editor, add the following text the file from exercise 2: 'next line added'.

4. Using both input and output redirection, copy the content the file 'chaucer' to a new file called 'chaucer-copy'.

What happens if you copy the contents of chaucer-copy to chaucer-copy, using output redirection?

5. Write a script called 'get-input' that read input from standard input three times in a row.

Supply the script with the input at startup of the script, using double input redirection.

6. Use an unnamed pipe to get a list of all the times the system was rebooted.

Level 2.

1. Create an empty file using output redirection.

```
# > emptyfile
```

2. Use two different methods to put the following text into a new file, without using an editor: 'straight from keyboard into file'.

```
(method 1)
# cat > file
straight into this file.
^D

(method 2)
# echo “straight from keyboard into file” > file
```

3. Again without using an editor, add the following text the file from exercise 2: 'next line added'.

```
(method 1)
# cat >> file
next line added
^D

(method 2)
# echo “next line added” >> file
```

4. Using both input and output redirection, copy the content the file 'chaucer' to a new file called 'chaucer-copy'.

```
# cat < chaucer > chaucer-copy
```

What happens if you copy the contents of chaucer-copy to chaucer-copy, using output redirection?

```
# cat chaucer-copy > chaucer-copy
```

cat: input/output files 'chaucer-copy' identical

As a result, the file is empty.

5. Write a script called '`get-input`' that reads input from standard input three times in a row.

```
# vi get-input
echo “give input”
read
echo “give input”
read
echo “give input”
read
:wq!
```

Supply the script with the input at startup of the script, using double input redirection.

```
# chmod +x
get-input
# ./get-input<<end
>yes
>no
>ok
>end
give input
give input
give input
```

6. Use an unnamed pipe to get a list of all the times the system was rebooted.

last | rebooted | more

Time Frame:
About one hour is needed for this lab.

Prerequisites:

Objectives:

Exercise Module 4. Arithmetic.

Level 1.

1. Echo the result of 100 divided by 3

2. Create a variable called myvar and value it with 56.
Now add 12 to this variable.

3. Subtract the variable myvar from myvar.
What is the result?

4. Write a script that asks for input.
If the input is an even number it returns “even” else it returns “odd”

5. Calculate the number of weeks left in this year.

6. If every page has 30 lines, how many pages of thirty lines does shortstories have?

Level 2.

1. Echo the result of 100 divided by 3

```
$ echo $(( 100 / 3 ))
33
```

2. Create a variable called myvar and value it with 56.
Now add 12 to this variable.

```
$ myvar="56"
$ echo $(( $myvar + 12 ))
68
```

3. Subtract the variable myvar from myvar.
What is the result?

```
$ echo $(( $myvar - $myvar ))
0
```

4. Write a script that asks for input.
If the input is an even number it returns “even” else it returns “odd”

```
echo -n "Enter a number > "
read number
echo "Number is $number"
  if [ $((number % 2)) -eq 0 ];
then
     echo "Number is even"
else
     echo "Number is odd"
fi
```

5. Calculate the number of weeks left in this year.

```
# echo $(( (365 - $(date +%j)) / 7 ))
11
```

6. If every page has 30 lines, how many pages of thirty lines does shortstories have?
7.

```
# echo $(( $(cat shortstories|wc -l) / 30 ))
22
```

Time Frame:
About one hour is needed for this lab.

Prerequisites:

Objectives:

Exercise Module 5. More parameters.

Level 1.

1. Write a script that does te following:

   Display the first and second positional parameter.

   Add the value of the first to the second and put then result in a variable called 'add'.

   Subtract the value of the second from the first and put the result in a variable called 'sub'.

   Multiply the first by the second and put the result in a variable called 'mul'.

   Divide the first by the second and put the result in a variable called 'div'.

   Display all the results on the screen.

2. Create a script that displays the first second and third positional parameter, the number of positional parameters and the value of all positional parameters.

3. Write a script that asks a filename with absolute pathname as input.
   Display the filename only.
   Display the pathname (without the filename)

4. What is the output of the following script when you call it with and without arguments?

```
$ cat pos1
#!/bin/bash
echo you entered the following argument:
echo ${1:-nothing}
```

Level 2.

1. Write a script that does te following:

   Display the first and second positional parameter.

   Add the value of the first to the second and put then result in a variable called 'add'.

   Subtract the value of the second from the first and put the result in a variable called 'sub'.

   Multiply the first by the second and put the result in a variable called 'mul'.

   Divide the first by the second and put the result in a variable called 'div'.

   Display all the results on the screen.

```
#!/bin/bash
#display values
echo -e  "\$1=$1"
echo -e "\$2=$2"
#calculate
let add=$1+$2
let sub=$1-$2
let mul=$1*$2
let div=$1/$2
#display results
echo $1 and $2 added : $add
echo $1 and $2 subtracted : $sub
echo $1 and $2 multiplied : $mul
echo $1 and $2 divided : $div
```

2. Create a script that displays the first second and third positional parameter, the number of positional parameters and the value of all positional parameters.

```
#!/bin/bash
echo “Called with $# parameters”
echo “The Shell Script Name is $0
echo “The first parameter is $1
echo “The second parameter is $2
echo “The third parameter is $3
echo “All parameters are $@”
```

3. Write a script that asks a filename with absolute pathname as input.
   Display the filename only.
   Display the pathname (without the filename)

```
#!/bin/bash
echo -n "enter absolute pathname plus filename : "
read name
echo "filename : ${name##*/}"
echo "pathname : ${name%/*}"
```

4. What is the output of the following script when you call it with and without arguments?

```
$ cat pos1
#!/bin/bash
echo you entered the following argument:
echo ${1:-nothing}

$ ./pos1 arg1
you entered the following argument:
arg1

$ ./pos1
you entered the following argument:
nothing
```

Exercise Module 6. Branches.

Level 1.

1. Write a script that checks whether any positional parameters were given when calling the script.

```
#!/bin/bash
if (( $# == 0 ))
then
          echo "no arguments"
else
          echo "$*"
fi
```

2. Write a script that searches for a file that is passed as an argument when running the script. If no argument is passed the script should ask for a filename.

```
#!/bin/bash
if (( $# == 0 ))
then
  echo -n "please enter filename : "
  read filename
  if [ -z $filename ]
  then
     echo "usage: search filename"
     exit 1
  else
     set $filename
  fi
 fi
#find filename
find / -name $1
```

3. Write a script that allows a user enter a filename. The script then checks what type of file it is.

```
#!/bin/bash
echo -n "enter filename : "
read filename
if [ -h $filename ]
then
 echo "symlink file"
elif  [ -f $filename ]
then
 echo "regular"
elif [ -b $filename ]
then
 echo "blockspecial"
elif [ -c $filename ]
then
 echo "characterspecial"
elif [ -d $filename ]
then
 echo "directory"
fi
```

4. Write a script using the '`case-construct`' that returns the text '`starting`' when the first argument is `start`, '`stopping`' when the first argument is `stop`, 'restarting' when the first argument is `restart` and '`error`' when no argument is passed when running the script.

```
#!/bin/bash
case $1 in
start) echo starting;;
stop) echo stopping;;
restart) echo restarting;;
*) echo error.
esac
```

Exercise Module 7. Loops.

Level 1.

1. Write a script that adds ".txt" to the name of every file in a particular directory, using a for-loop.

```
 #!/bin/bash
for i in `ls`
do
  mv ${i} ${i}.txt
  echo "changed ${i}"
done
```

2. Write a script that create a 100 files with the name file and a sequence number. So you should have file: file1 up to file100. The files are empty.
   Use a for-loop and the 'seq' command.

```
#!/bin/bash
for i in `seq 1 100`
do
  touch file${i}

done
```

3. Write a script that asks the user to type 'yes' 'no' or 'quit'.
   The script should give an error message if another value is typed.
   If the user types 'yes', you respond with yes, if he types 'no' you reply with no. If he types 'quit', the script stops with exit 0.

   Use a while-loop with either an if statement or a case statement.

```
 #!/bin/bash
 while true
do
echo -n "yes, no or quit "
read input
case  $input in
yes) echo yes;;
no)  echo no;;
quit) exit 0;;
*) echo "incorrect, try again";;
esac
done
```

4. Write a script that takes arguments when it is invoked (positional parameters). If the argument is a filename it copies the file to the /tmp/ directory.

   Use a for-loop with the shift command, and check whether the file exists before you copy it. And if no arguments are passed, an error is given and the script exits with return-code 1.

```
if [[ $# -lt 1 ]]
then
  echo "no arguments given"
  exit
fi

for i in $*
do
 if [[ -f $i ]]
 then
    cp $i /tmp
    echo $i copied
    shift
 fi
done
```