LPI 107.2 - Customize or write simple scripts

Curs 2021 - 2022

ASIX M01-ISO 105 Shells and shell scripting

Customize or write simple scripts

Description

Key concepts:

- ❏ Use standard sh syntax (loops, tests).
- ❏ Use command substitution.
- ❏ Test return values for success or failure or other information provided by a command.
- ❏ Execute chained commands.
- ❏ Perform conditional mailing to the superuser.
- ❏ Correctly select the script interpreter through the shebang (#!) line.
- ❏ Manage the location, ownership, execution and suid-rights of scripts.

Commands and files:

- ❏ for
- ❏ while
- ❏ test
- ❏ if
- ❏ read
- ❏ seq
- ❏ exec
- ❏ ||
- ❏ &&

Shell Scripts Basics

The process to create a shell script consists of:

- Create the script using your favorite editor. Name the file script-name.sh.
- Write the Shebang in the first line of the file, indicating that's a /bin/bash shell script.
  `#! /bin/bash`
- Change the file permissions activating the execution bit:
  `chmod +x script-name.sh`

```
#! /bin/bash
# hello.sh script example
#
echo "Wellcome to LPI"
echo "this is a message"
name="pere pou prat"
age=25
echo "$name $age"
echo "this is a literal $name $age"
```

```
uname -a
uptime
echo $SHELL
echo $SHLVL
echo "4*32 is: $((4*32))"
echo $((age*2))
#read data1 data2
# echo -e "data1: $data1\ndata2: $data2"
exit 0
```

There are many ways to execute the script:

- relative path
- absolute path
- place the script in the PATH
- using a subshell with bash
- using source (.)

Execute using a relative path

```
#1
$ vim hello.sh
$ chmod +x hello.sh
$ ./hello.sh
Wellcome to LPI
this is a message
pere pou prat 25
this is a literal pere pou prat 25
Linux localhost.localdomain 5.11.22-100.fc32.x86_64 #1 SMP Wed May 19 18:58:25 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
 16:33:04 up  3:22,  1 user,  load average: 0.47, 0.56, 0.64
/bin/bash
2
4*32 is: 128
50
```

Execute using and absolute path:

```
#2
$ /var/tmp/pue/hello.sh
Wellcome to LPI
this is a message
pere pou prat 25
this is a literal pere pou prat 25
Linux localhost.localdomain 5.11.22-100.fc32.x86_64 #1 SMP Wed May 19 18:58:25 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
 16:37:18 up  3:26,  1 user,  load average: 0.39, 0.46, 0.58
/bin/bash
2
4*32 is: 128
50
```

Execute placing the file in the PATH

```
#3
[root@localhost ~]# cp /var/tmp/pue/hello.sh /usr/bin/
$ which hello.sh
/usr/bin/hello.sh
$ hello.sh
Wellcome to LPI
this is a message
pere pou prat 25
this is a literal pere pou prat 25
Linux localhost.localdomain 5.11.22-100.fc32.x86_64 #1 SMP Wed May 19 18:58:25 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
 16:40:24 up  3:29,  1 user,  load average: 0.14, 0.34, 0.51
```

```
/bin/bash
2
4*32 is: 128
50
```

```
# rm /usr/bin/hello.sh
```

Execute using a subshell bash

```
#4
$ bash hello.sh
Wellcome to LPI
this is a message
pere pou prat 25
this is a literal pere pou prat 25
Linux localhost.localdomain 5.11.22-100.fc32.x86_64 #1 SMP Wed May 19 18:58:25 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
 16:41:03 up  3:30,  1 user,  load average: 0.18, 0.32, 0.50
/bin/bash
2
4*32 is: 128
50
```

Execute in the actual shell using source (.)

```
#5
$ source hello.sh
*note* the final exit closes the session
```

```
$ . hello.sh
*note* the final exit closes the session
```

```
*note* without the final exit

$ . hello.sh
Wellcome to LPI
this is a message
pere pou prat 25
this is a literal pere pou prat 25
Linux localhost.localdomain 5.11.22-100.fc32.x86_64 #1 SMP Wed May 19 18:58:25 UTC 2021
x86_64 x86_64 x86_64 GNU/Linux
 16:44:57 up  3:34,  1 user,  load average: 0.15, 0.23, 0.42
/bin/bash
1
4*32 is: 128
50
```

Command Substitution and expansions

Command substitution allows the output of a command to replace the command name. There are two forms:

- $(command) [preferred syntax]
- `command` [not recommended]

Bash performs the expansion by executing command in a subshell environment and replacing the command substitution with the standard output of the ommand, with any trailing newlines deleted. Embedded newlines are not deleted, but they may be removed during word splitting.

Command substitution us mainly used for:

- redirect the output of a command into a variable.
- insert the output of a command as another command's argument

```
#6
$ list_users=$(grep ":/bin/bash$" /etc/passwd | cut -d: -f1)
$ echo $list_users
root guest ecanet
```

```
$ GNAME=users
$ gid=$(grep "^$GNAME:" /etc/group | cut -d: -f3)
$ echo $gid
100
```

```
$ nobash=$(grep -v ":/bin/bash$" /etc/passwd | wc -l)
$ echo $nobash
49
```

```
#7
$ echo "today is $(date)"
today is Tue 02 Nov 2021 05:03:55 PM CET
```

```
$ rpm -qi $(rpm -qf /etc/passwd)
Name        : setup
Version     : 2.13.6
Release     : 2.fc32
Architecture: noarch
Install Date: Wed 07 Apr 2021 11:53:32 PM CEST
Group       : System Environment/Base
Size        : 726694
License     : Public Domain
Signature   : RSA/SHA256, Fri 31 Jan 2020 02:24:52 AM CET, Key ID 6c13026d12c944d0
Source RPM  : setup-2.13.6-2.fc32.src.rpm
Build Date  : Fri 31 Jan 2020 02:21:21 AM CET
Build Host  : buildvm-aarch64-21.arm.fedoraproject.org
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : https://pagure.io/setup/
Bug URL     : https://bugz.fedoraproject.org/setup
Summary     : A set of system configuration and setup files
Description :
The setup package contains a set of important system configuration and
setup files, such as passwd, group, and profile.
```

Arithmetic Expansion

- $((expression))

```
#8
$ echo $((2*4))
8

$ num=15
$ echo $((num*8))
120
```

Statements

- read
- if
- test
- for
- while
- exec

Read statement

The read statement can be used to gather information from the user who is running the script. The user will be presented with a blinking cursor that will accept keyboard input and place what the user types into one or more variables.

```
read var
read var,var…
read -p “message” var,var...
```

```
#9
$ read -p "enter your name:" name
enter your name:pere

$ echo $name
pere
```

```
$ read -p "enter your name and age: " name age
enter your name and age: pere 24

$ echo $name $age
pere 24
```

```
$ read -p "enter your name and age: " name age
enter your name and age: pere pou prat 24

$ echo $name $age
pere pou prat 24

$ echo $name
pere

$ echo $age
pou prat 24
```

Test statement

The test command can perform numeric comparisons, string comparisons, and file testing operations:

- If two string variables or values match (or don’t match)

- If two numeric variables or values match (or don't match)
- The status of files (if file exists, if file is a directory, etc.)
- If a command completes successfully

test can be used alone and in if statements. Adopts two syntaxes (make sure to place spaces around the brackets):

- test 5 -lt 10
- [5 -lt 10]

Test numeric comparators (check man test for more information):

```
      INTEGER1 -eq INTEGER2
              INTEGER1 is equal to INTEGER2

       INTEGER1 -ge INTEGER2
              INTEGER1 is greater than or equal to INTEGER2

       INTEGER1 -gt INTEGER2
              INTEGER1 is greater than INTEGER2

       INTEGER1 -le INTEGER2
              INTEGER1 is less than or equal to INTEGER2

       INTEGER1 -lt INTEGER2
              INTEGER1 is less than INTEGER2

       INTEGER1 -ne INTEGER2
              INTEGER1 is not equal to INTEGER2
```

Test string comparators:

```
       -n STRING
              the length of STRING is nonzero

       STRING equivalent to -n STRING

       -z STRING
              the length of STRING is zero

       STRING1 = STRING2
              the strings are equal

       STRING1 != STRING2
              the strings are not equal
```

Test file comparators:

```
       -b FILE              FILE exists and is block special
       -c FILE              FILE exists and is character special
       -d FILE              FILE exists and is a directory
       -e FILE              FILE exists
       -f FILE              FILE exists and is a regular file
       -g FILE              FILE exists and is set-group-ID
       -G FILE              FILE exists and is owned by the effective group ID
       -h FILE              FILE exists and is a symbolic link (same as -L)
       -k FILE              FILE exists and has its sticky bit set
       -L FILE              FILE exists and is a symbolic link (same as -h)
       -N FILE              FILE exists and has been modified since it was last
read
       -O FILE              FILE exists and is owned by the effective user ID
       -p FILE              FILE exists and is a named pipe
       -r FILE              FILE exists and read permission is granted
       -s FILE              FILE exists and has a size greater than zero
       -S FILE              FILE exists and is a socket
       -t FD  file descriptor FD is opened on a terminal
       -u FILE              FILE exists and its set-user-ID bit is set
       -w FILE              FILE exists and write permission is granted
```

```
       -x FILE              FILE exists and execute (or search) permission is
granted
```

Test logical operations

```
       ( EXPRESSION )
              EXPRESSION is true

       ! EXPRESSION
              EXPRESSION is false

       EXPRESSION1 -a EXPRESSION2
              both EXPRESSION1 and EXPRESSION2 are true

       EXPRESSION1 -o EXPRESSION2
              either EXPRESSION1 or EXPRESSION2 is true
```

Check the test man or help page for all the comparassions.

```
#10
$ test 5 -lt 10

$ test 5 -lt 10 && echo "ok"
ok

$ test 5 -ge 10 && echo "ok"

$ test 10 -ge 10 && echo "ok"
ok

$ test 10 -eq 10 && echo "ok"
ok
```

```
$ test "pere" != "marta" && echo "differ"
differ

$ name="pere"
$ test "pere"=$NAME && echo "ok"
ok
```

```
$ test -d /tmp && echo "ok"
ok

$ test -f /etc/fstab && echo "ok"
ok


$ test -c /dev/tty0 && echo "ok"
ok

$ test -r /etc/fstab && echo "ok"
ok

$ test -x /etc/fstab && echo "ok"

$ test -w /etc/fstab && echo "ok"
```

```
#11

$ num=5
$ if [ $num -lt 10 ]; then
>   echo "less"
> fi
less
```

```
$ if [ -d /tmp ]; then
>   echo "is directory"
> fi
```

```
is directory
```

if statement

- simple if
- if/else
- if/elif else

```
if condition
then
  actions
fi
```

```
if condition
then
  actions
else
  actions
fi
```

```
if condition
the
  actions
elif condition
then
  actions
else
  actions
fi
```

```
#12

if [ -e /etc/fstab ]; then
  echo "file /etc/fstab exists"
fi
```

```
if [ -w /etc/fstab ]; then
  echo "file /etc/fstab is writable"
else
  echo "file /etc/fstab is not writable"
fi
```

```
if [ -w /etc/fstab ]; then
  echo "file /etc/fstab is writable"
elif [ -x /etc/fstab ];then
  echo "file /etc/fstab is executable"
elif [ -r /etc/fstab ]; then
  echo "file /etc/fstab is readable"
else
  echo "file /etc/fstab is not w x r"
fi
```

```
$ bash if.sh
file /etc/fstab exists
file /etc/fstab is not writable
file /etc/fstab is readable
```

Return Values

Every command has a return or exit status value that can be used to determine if the command succeeded or failed. This return value is a number that can only be an integer value from 0 to 255. A value of 0 means the command executed successfully while a positive integer value means the command failed. To see the exit status of a command, view the **$?** variable.

```
#13
$ date
Tue 02 Nov 2021 05:49:32 PM CET
$ echo $?
0

$ dateeee
bash: dateeee: command not found...
$ echo $?
127
```

```
$ grep "^root" /etc/passwd
root:x:0:0:root:/root:/bin/bash
$ echo $?
0

$ grep "^unknown" /etc/passwd
$ echo $?
1
```

```
$ mkdir /tmp/dir
$ echo $?
0

$ mkdir /tmp/dir
mkdir: cannot create directory ‘/tmp/dir’: File exists
$ echo $?
1
```

```
#14
$ cat userdel-check.sh
#! /bin/bash
# example use of  $?
#
userdel -r pere
if [ $? -eq 0 ]; then
  echo "user pere deleted"
else
  echo "error deleting user pere"
fi

$ bash userdel-check.sh
userdel: user 'pere' does not exist
error deleting user pere
```

While statement

The while statement is used to determine if a condition is true or false; if it is true, then a series of actions take place, and the condition is checked again. If the condition is false, then

no action takes place, and the program continues. The while statement, along with the block of statements to be executed when the condition is true, is also known as a while loop.

```
while condition
do
  actions
done
```

```
#15
$ cat while.sh
#! /bin/bash
# show from [1-10]
MAX=10
num=1
while [ $num -le $MAX ]
do
  echo "$num"
  ((num++))
done
exit 0

$ bash while.sh
1
2
3
4
5
6
7
8
9
10
```

```
#16
$ cat while-num-stdin.sh
#! /bin/bash
# numerar stdin  línia a línia
num=1
while read -r line
do
  echo "$num: $line"
  ((num++))
done
exit 0

$ cat > noms.txt
pere
pau
marta
anna
^d

$ bash while-num-stdin.sh  < noms.txt
1: pere
2: pau
3: marta
4: anna
```

For Statement

The for statement iterates for each element in a list.

```
for var in list-of-values
do
  action
done
```

```
#17
$ cat for.sh
#! /bin/bash
# for example
for name in pere anna marta
do
  echo useradd $name
done

$ bash for.sh
useradd pere
useradd anna
useradd marta
```

```
#18
$ cat for-args.sh
#! /bin/bash
# iterate args
num=1
for arg in $*
do
 echo "$num: $arg"
 ((num++))
done

$ bash for-args.sh pere anna ramon marta
1: pere
2: anna
3: ramon
4: marta
```

Seq Statement

The seq statement generates a list of integer values.

```
seq initial-value step final-value
```

```
#19
$ seq 5 10
5
6
7
8
9
10

$ seq 1 2 10
1
3
5
7
9

$ seq 2 3 10
2
5
8
```

The exec command

Recall that a command is a program that, in most cases, creates a process when executed and then returns to the current process, the shell. However, the exec command behaves differently. The exec command replaces the current process with a specified command. If no command is specified, the exec command spawns a new shell process and returns a shell prompt.

The exec command is also commonly used in scripts to start a new shell with a clean environment as read by the shell startup files.

```
exec command
```

Positional Parameters

Special parameters:

- $#
- $*
- $@
- $1 $2 … ${10} ${11}...

```
#20
$ cat args.sh
#! /bin/bash
# argument example
echo '$*:' $*
echo '$@:' $@
echo '$#:' $#
echo '$0:' $0
echo '$1:' $1
echo '$2:' $2
echo '$9:' $9
echo '$10:' ${10}
echo '$11:' ${11}

$ bash args.sh  aa bb cc dd ee ff gg hh ii jj kk ll
$*: aa bb cc dd ee ff gg hh ii jj kk ll
$@: aa bb cc dd ee ff gg hh ii jj kk ll
$#: 12
$0: args.sh
$1: aa
$2: bb
$9: ii
$10: jj
$11: kk
```

```
#21
$ cat for-args.sh
#! /bin/bash
# iterate args
num=1
```

```
for arg in $*
do
 echo "$num: $arg"
 ((num++))
done

$ bash for-args.sh pere anna ramon marta
1: pere
2: anna
3: ramon
4: marta
```

Functions

```
function name(){
  actions
  return value
}
```

```
#21
$ cat function.sh
#! /bin/bash
# function example
function suma(){
  a=$1
  b=$2
  suma=$((a+b))
  echo $suma
  return 0
}

function multiplica(){
  echo $(($1*$2))
  return 0
}

# main
suma 5 3
multiplica 4 5

$ bash function.sh
8
20
```

Example Exercises

1. Create a hello.sh shell program showing your name, uid, uptime and date.
2. Execute the hello.sh script:
    a. using a relative path
    b. using an absolute path
    c. in a subshell using bash
    d. in the shell using source
3. Using command substitution assign to the variables the appropriate value:
    a. hostaname ← the name of the host
    b. date ← the date in format dd-mm-aaaa

    c. five ← the first five sorted login names in the /etc/passwd
    d. shells ← the different shells assigned to the users, found in /etc/passwd.
4. Write and execute the next test expressions:
    a. file (element) not exists
    b. is not a regular file.
    c. num is not equal to 10
    d. num is less or equal to 100
    e. name is not pere, not marta not anna
    f. num is 10 or 15 or 20.
5. Write a program that creates a directory (argument) and shows a message if the directory can be created and another error message if it can not.
6. Write a program to determine if an student grade is: suspès, aprovat, notable, excel·lent.
7. Write a program to show the arguments in one line each and numbered.
8. Write a program to show the numbers form a to b (both arguments).

9. Realitza els exercicis indicats a:
10. Realitza els exercicis del Question-Topics 105.2.