Chapter 1

These are some exercises to warm you up for the next chapter:

1. Where is the bash program located on your system?

Standard location of the Bourne Again shell is installed in /bin.

In /bin/bash

2. Use the --version option to find out which version you are running.

consola: bash --version

GNU bash, versión 4.4.19(1)-release (x86_64-pc-linux-gnu)

Copyright (C) 2016 Free Software Foundation, Inc.

Licencia GPLv3+: GPL de GNU versión 3 o posterior <http://gnu.org/licenses/gpl.html>

3.Which shell configuration files are read when you login to your system using the graphical user interface and then opening a terminal window?

`~/.bashrc`

4. Are the following shells interactive shells? Are they login shells?

- A shell opened by clicking on the background of your graphical desktop, selecting "Terminal" or such from a menu. A: Yes, No
- A shell that you get after issuing the command ssh localhost. A: Yes, Yes.
- A shell that you get when logging in to the console in text mode.
- A shell obtained by the command xterm &.
- A shell opened by the mysystem.sh script.
- A shell that you get on a remote host, for which you didn't have to give the login and/or password because you use SSH and maybe SSH keys.

5. Can you explain why bash does not exit when you type Ctrl+C on the command line?

Because SIGINT is caught and handled properly, so no program termination is done.

Es una opcion valida para el modo interactivo. In the absence of traps, SIGINT is caught and handled. Thus, typing Ctrl+C, for example, will not quit your interactive shell.

6. Display directory stack content.

`echo $DIRSTACK`. This will be initially empty. To fill the stack, you have to use `pushd` and `popd` to traverse between directories.

The directory stack is a list of recently-visited directories. The pushd built-in adds directories to the stack as it changes the current directory, and the popd built-in removes specified directories from the stack and changes the current directory to the directory removed. Content can be displayed issuing the dirs command or by checking the content of the DIRSTACK variable.

7. If it is not yet the case, set your prompt so that it displays your location in the file system hierarchy, for instance add this line to ~/.bashrc:

export PS1="\u@\h \w> "

literal es copiar ese comando en terminal, y va a a cambiar el formato de username@hostname por esa sesión por uno que te muestra el directorio en el que estás actualmente para hacer permanente este cambio hay que modificar el valor de PS1 en bashrc

8. Display hashed commands for your current shell session.

```
hits    command
1       /bin/rm
1       /bin/mkdir
1       /usr/bin/gedit
1       /bin/ls
```

9. How many processes are currently running on your system? Use ps and wc, the first line of output of ps is not a process!

I use ps to find the processes, then redirect its output to the wc command with -l to count lines : `ps | wc -l`

10. How to display the system hostname? Only the name, nothing more!

```
echo $HOSTNAME
```

Chapter 2

This exercise will help you to create your first script.

1. Write a script using your favorite editor. The script should display the path to your homedirectory and the terminal type that you are using. Additionally it shows all the services started up in runlevel 3 on your system. (hint: use HOME, TERM and ls /etc/rc3.d/S*)

2. Add comments in your script.

3. Add information for the users of your script.

```
#!/bin/bash

clear

echo "The script starts now."
echo

echo "Hi, $USER, your home folder is at $HOME" # to print the home, we need to use $HOME
echo

echo "You are using $TERM" # $TERM for current terminal emulator
echo

echo "Services started up in runlevel 3"
ls /etc/rc3.d/S*    # prints the services at runlevel 3
echo
```

4. Change permissions on your script so that you can run it.

`chmod +x script.sh` and then `./script.sh`

5. Run the script in normal mode and in debug mode. It should run without errors.

Normal mode: `./script.sh` Debug mode : `bash -x script.sh`

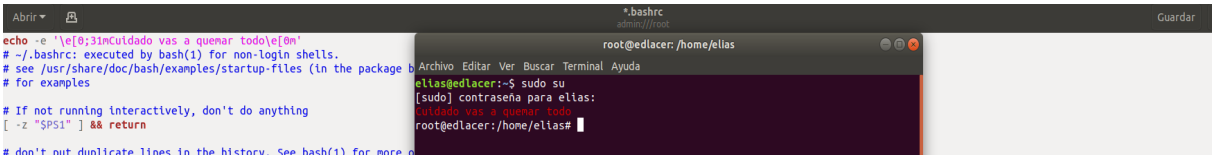
6. Make errors in your script: see what happens if you misspell commands, if you leave out the first line or put something unintelligible there, or if you misspell shell

variable names or write them in lower case characters after they have been declared in capitals. Check what the debug comments say about this.
1. Al poner value en minusculas, y ejecutar el comando, no me devuelve el valor por pantalla, pero al ejecutar con el debugger
`bash -x script.sh`
Y cuando se ejecuta: echo "This is a number: $VALUE"
No me muestra el valor de value, y tampoco lo hace al imprimir por pantalla.
2. Tambien, si pongo en una linea ehco en vez de echo, me muestra que la linea se ejecuta, pero tiene esta respuesta.
script1.sh: línea 10: ehco: orden no encontrada

Chapter 3

For this exercise, you will need to read the useradd man pages, because we are going to use the /etc/skel directory to hold default shell configuration files, which are copied to the home directory of each newly added user.
First we will do some general exercises on setting and displaying variables.
1. Create 3 variables, VAR1, VAR2 and VAR3; initialize them to hold the values "thirteen", "13" and "Happy Birthday" respectively.
VAR1="thirteen"
VAR2="13"
VAR3="Happy Birthday"

2. Display the values of all three variables. echo $VAR{1,2,3}
3. Are these local or global variables?
local pq se tiene que definir explícitamente como global, y al definir nomas se define como local
4. Remove VAR3. unset VAR3
5. Can you see the two remaining variables in a new terminal window?
no pq son locales jaja xd
6. Edit /etc/profile so that all users are greeted upon login (test this).
echo “Hola!” en el archivo /etc/profile Cambiar permiso si hace falta con chmod
probar con bash -login
7. For the root account, set the prompt to something like "Danger!! root is doing stuff in \w", preferably in a bright color such as red or pink or in reverse video mode.



Modificar el archivo bashrc de root ver primera linea del gedit
y el resultado en la terminal
para cambiar de color todo sacado de google
8. Make sure that newly created users also get a nice personalized prompt which informs them on which system in which directory they are working. Test your changes by adding a new user and logging in as that user.

9. Write a script in which you assign two integer values to two variables. The script should calculate the surface of a rectangle which has these proportions. It should be aired with comments and generate elegant output.

```
#!/bin/bash

#Asignación de variables
VAR1=4
VAR2=5

echo "El rectángulo con largo y ancho igual a $[$VAR1>=$VAR2?$VAR1:$VAR2] y $[$VAR1<=$VAR2?$VAR1:$VAR2] respectivamente tiene por área-> $[VAR1*VAR2]"
#Impresión de las dimensiones del rectángulo y su área
```

Chapter 4

These exercises will help you master regular expressions.

1. Display a list of all the users on your system who log in with the Bash shell as a default.

```
grep bash /etc/passwd | cut -d: -f1
```

2. From the /etc/group directory, display all lines starting with the string "daemon".

```
grep "^daemon" /etc/group
```

3. Print all the lines from the same file that don't contain the string.

```
grep --invert-match "^daemon"
```

4. Display localhost information from the /etc/hosts file, display the line number(s) matching the search string and count the number of occurrences of the string.

info with line numbers:

```
grep -n localhost /etc/hosts
```

number of occurrences of string:

```
grep -o localhost /etc/hosts |wc -w
```

5. Display a list of /usr/share/doc subdirectories containing information about shells.

```
cd /usr/share/doc
for file in *
do
        if grep "$file" /etc/shells &gt; /dev/null
        then
```

```
                echo "$file"
        fi
done
```

6. How many README files do these subdirectories contain? Don't count anything in the form of "README.a_string".

```
sum=0
for file in *
do
        if grep "$file" /etc/shells &gt; /dev/null
        then
                sum=`expr $sum + $(find "$file" -name README |wc -l)`
        fi
done
echo $sum
```

7. Make a list of files in your home directory that were changed less that 10 hours ago, using grep, but leave out directories.

```
find . -mtime 0.42 -type f
for file in *
do
        if [ `stat -c %Y "$file"` -gt `expr $(date "+%s") - 36000` ] &amp;amp;&amp;amp; [ -f "$file" ]
        then
                echo "$file"
        fi
done
```

8. Put these commands in a shell script that will generate comprehensible output.
9. Can you find an alternative for wc -l, using grep?

```
grep -c ".*" test
```

10. Using the file system table (/etc/fstab for instance), list local disk devices.

```
grep "^/dev" /etc/fstab | awk '{print $1}'
grep -o "^/dev[^[:space:]]*" /etc/fstab
```

11. Make a script that checks whether a user exists in /etc/passwd. For now, you can specify the user name in the script, you don't have to work with arguments and conditionals at this stage.

```
user=rwh
if grep "^$user:" /etc/passwd &gt; /dev/null
then
        echo user $user exists in the fstab
else
```

```
        echo user $user does not exist in the fstab
fi
```

(Optativo)

```
#! /bin/bash

VAR1=PablitoPintos
#Verificar si un usuario existe
echo "Verificando si el usuario $VAR1 existe.."
echo "a continuación se imprimirá 1 si existe, 0 caso contrario:"
cat /etc/passwd | grep  ^"$VAR1:" -c
#se ingresa al archivo con los datos de usuarios del sistema y se busca el primer patrón de cada línea que coincida con "nombre:",":" evita posibles conflictos
```

12. Display configuration files in /etc that contain numbers in their names.

```
cd /etc
ls |grep [0-9]
ls |grep "[[:digit:]]"
```

(Optativo)

```
ls -R /etc/ | grep [0-9] | grep '\.conf$'
```

jjiijiji hice toro este llanto por nara / jajaajaj

Chapter 5

These exercises are meant to further demonstrate what sed can do.

1. Print a list of files in your scripts directory, ending in ".sh". Mind that you might have to unalias ls. Put the result in a temporary file.

```
ls -R /home/alecsi/Escritorio/scrip | sed -n '/\.sh$/p'>temp1.txt
```

2. Make a list of files in /usr/bin that have the letter "a" as the second character. Put the result in a temporary file.

```
ls /usr/bin | sed -n  '/^.a/p' >temp2.txt
```

//

Proporcione un script denominado testdaemon.sh tal que el mismo compruebe si un servicio (demonio) cuyo nombre es recibido como argumento se encuentra o no en ejecución.

```
#!/bin/bash

VAR1=$(ps -e | grep $1 -c)

if (( $VAR1 >= 1 ))
then
 echo "Se encuentra en ejecucion"
else
 echo "No se encuentra en ejecucion"
fi
```

//

3. Delete the first 3 lines of each temporary file.

```
sed '1,3d' temp1.txt > temp.txt

mv temp.txt temp1.txt
```

4. Print to standard output only the lines containing the pattern "an".

```
sed -n '/an/p' temp2.txt
```

5. Create a file holding sed commands to perform the previous two tasks. Add an extra command to this file that adds a string like "* This might have something to do with man and man pages ***" in the line preceding every occurence of the string "man". Check the results.**

 # comando (supongo que pide meter en un script sh nomas aca)

```
#! /bin/bash

#Ejercicio 3
echo '-------------ejercicio 3---------------'
sed '1,3d' temp1.txt > temp.txt
```

```
mv temp.txt temp1.txt

#Ejercicio 4
echo '-------------ejercicio 4----------------'
sed -n '/an/p' temp2.txt

#Ejercicio 5
echo '-------------ejercicio 5-----------------'
sed '/man/i ***Esto podria tener algo que ver con las paginas man y
man***' temp2.txt
```

6. A long listing of the root directory, /, is used for input. Create a file holding sed commands that check for symbolic links and plain files. If a file is a symbolic link, precede it with a line like "--This is a symlink--". If the file is a plain file, add a string on the same line, adding a comment like "<--- this is a plain file".
 # comando (supongo que pide meter en un script sh nomas aca)

```
#! /bin/bash

#Ejercicio 6
echo '-------------ejercicio 6-----------------'

ls -l / |sed -e 's/^/>/' -e 's/^>l/--Este es un enlace simbolico-- /'
-e 's/^>-/--Este es un archivo simple-- /' -e 's/^>d/--Este es un
enlace simple-- /' |awk '{print $1,$2,$3,$4,$5" \t" $14}'
```

7. Create a script that shows lines containing trailing white spaces from a file. This script should use a sed script and show sensible information to the user.

 # comando (ya no puedo suponer porque dice script sed, bajon)

Chapter 6

These are some practical examples where awk can be useful.
1. For the first exercise, your input is lines in the following form:
Username:Firstname:Lastname:Telephone number
Make an awk script that will convert such a line to an LDAP record in this format:

dn: uid=Username, dc=example, dc=com
cn: Firstname Lastname
sn: Lastname
telephoneNumber: Telephone number

#meter en un archivo.awk

BEGIN { FS=":"}

{print "dn: uid="$1 ", dc=example, dc=com"}
{print "n:" $2,$3}
{print "sn:" $3}
{print "telephoneNumber:" $4}

Create a file containing a couple of test records and check.

2. Create a Bash script using awk and standard UNIX commands that will show the top three users of disk space in the /home file system (if you don't have the directory holding the homes on a separate partition, make the script for the / partition; this is present on every UNIX system). First, execute the commands from the command line. Then put them in a script. The script should create sensible output (sensible as in readable by the boss). If everything proves to work, have the script email its results to you (use for instance mail -s Disk space usage <you@your_comp> < result). If the quota daemon is running, use that information; if not, use find.
Link:
https://www.cyberciti.biz/tips/shell-script-to-watch-the-disk-space.html
https://bash.cyberciti.biz/monitoring/shell-script-monitor-unix-linux-diskspace/

#Shell script to monitor or watch the disk space and send an email alert if the (free available) percentage of space is >= 90%

```
#!/bin/sh
# Shell script to monitor or watch the disk space
# It will send an email to $ADMIN, if the (free available) percentage
# of space is >= 90%
# ------------------------------------------------------------------------
# Copyright (c) 2005 nixCraft project <http://cyberciti.biz/fb/>
# This script is licensed under GNU GPL version 2.0 or above
# ------------------------------------------------------------------------
# This script is part of nixCraft shell script collection (NSSC)
# Visit http://bash.cyberciti.biz/ for more information.
# ----------------------------------------------------------------------
# Linux shell script to watch disk space (should work on other UNIX oses )
```

```
# SEE URL:
http://www.cyberciti.biz/tips/shell-script-to-watch-the-disk-space.html
# set admin email so that you can get email
ADMIN="me@somewher.com"
# set alert level 90% is default
ALERT=90
df -H | grep -vE '^Filesystem|tmpfs|cdrom' | awk '{ print $5 " " $1 }' |
while read output;
do
  #echo $output
  usep=$(echo $output | awk '{ print $1}' | cut -d'%' -f1  )
  partition=$(echo $output | awk '{ print $2 }' )
  if [ $usep -ge $ALERT ]; then
    echo "Running out of space \"$partition ($usep%)\" on $(hostname) as on
$(date)" |
     mail -s "Alert: Almost out of disk space $usep" $ADMIN
  fi
done
```

3. Create XML-style output from a Tab-separated list in the following form:

Meaning very long line with a lot of description
meaning another long line
othermeaning more longline
testmeaning
looooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooong line,
The output should read:
<row>
<entry>Meaning</entry>
<entry>
very long line
</entry>
</row>
<row>
<entry>meaning</entry>
<entry>
long line
</entry>
</row>
<row>
<entryothermeaning</entry>
<entry>
more longline
</entry>
</row>
<row>

<entrytestmeaning</entry>
<entry>
looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
ooooooong line, but i mean really </entry>
</row>
Additionally, if you know anything about XML, write a BEGIN and END script to complete the table. Or do it in HTML.

```
#meter en un archivo.awk
BEGIN { FS="\t"}

{print "<row>"}
{i=1}
# no puedo usar for porque la variable $# no funka en awk y me da ansiedad
{while($i!="") {

	print "<entry>"
	print $i
	print "</entry>"
	i++
}
}

{print "</row>"}

#------------------------------Otra opcion---------------
#en ejercicio.awk

BEGIN {FS="\t"}

{ print "<row>" }
{ print "<entry>" $1 "</entry>"}
{ print "<entry>\n" $2 "\n</entry>"}
{ print "</row"}

#En prueba.txt
Meaning	very long line with a lot of description
meaning	another long line
othermeaning	more longline
testmeaning	loooooooooooooooooooooooooooooooooooooooong
```

Chapter 7

Here are some ideas to get you started using if in scripts:
1. Use an if/then/elif/else construct that prints information about the current month. The script should print the number of days in this month, and give information about leap years if the current month is February.

```
#!/bin/bash

# ojo, ejecutar usando ./ (chmod u+x) ya que sh no posee integrado
funciones que utiliza este scrip (())

year=$(date +%Y)
mes=$(date +%m)
Mes=$(date +%B)

# verificar que mes es
if (( $mes == 1 )) ||  (( $mes == 3 )) || (( $mes == 5 )) || (( $mes ==
7 )) || (( $mes == 8 )) || (( $mes == 10 )) || (( $mes == 12 ));then
       echo "$Mes posee 31 dias"
#en caso de no pertenecer a los citados anteriormente y no ser febrero
elif (($mes != 2 ));then
       echo "$Mes posee 30 dias"
#en caso de ser febrero se necesitara saber si es bisiesto o no
elif (( $year % 400 == 0)) || ( (($year % 4 == 0)) && (( $year % 100 !=
0)) ); then
       echo "$Mes posee 29 dias, es un año bisiesto!"
#en caso de no ser bisiesto tendra 28 dias
else
   echo "$Mes posee 28 dias, el año no es bisiesto"

fi


////////////////////////////////////////////////////////////////
```

```
#!/bin/bash
 year=$(date +%Y)
mes=$(date +%m)
Mes=$(date +%B)
if (( $mes == 1 )) ||  (( $mes == 3 )) || (( $mes == 5 )) || (( $mes ==
7 )) || (( $mes == 8 )) || (( $mes == 10 )) || (( $mes == 12 ));then
      echo "$Mes posee 31 dias"
#en caso de no pertenecer a los citados anteriormente y no ser febrero
elif (($mes != 2 ));then
```

```
      echo "$Mes posee 30 dias"
#en caso de ser febrero se necesitara saber si es bisiesto o no
elif (( $year % 400 == 0)) || ( (($year % 4 == 0)) && (( $year % 100 != 0)) ); then
      echo "$Mes posee 29 dias"
#en caso de no ser bisiesto tendra 28 dias
else
  echo "$Mes posee 28 dias"
fi
if (( $year % 400 == 0)) || ( (($year % 4 == 0)) && (( $year % 100 != 0)) ); then
      echo "El anho es bisiesto"
else
 echo "El anho no es bisiesto"
fi
////////////////////////////////////////////////////////////////////////
```

2. Do the same, using a case statement and an alternative use of the date command.

```
#!/bin/bash

# ojo, ejecutar usando ./ (chmod u+x) ya que sh no posee integrado funciones que utiliza este scrip (())

year=$(date +%Y)
mes=$(date +%m)
Mes=$(date +%B)

case $Mes in

   enero)
       echo "$Mes posee 31 dias"
       ;;
   febrero)
       if (( $year % 400 == 0)) || ( (($year % 4 == 0)) && (( $year % 100 != 0)) ); then
           echo "$Mes posee 29 dias, es un año bisiesto!"
       #en caso de no ser bisiesto tendra 28 dias
       else
            echo "$Mes posee 28 dias, el año no es bisiesto"
       fi
```

```
        ;;
    marzo)
        echo "$Mes posee 31 dias"
        ;;
    abril)
        echo "$Mes posee 30 dias"
        ;;
    mayo)
        echo "$Mes posee 31 dias"
        ;;
    junio)
        echo "$Mes posee 30 dias"
        ;;
    julio)
        echo "$Mes posee 31 dias"
        ;;
    agosto)
        echo "$Mes posee 31 dias"
        ;;
    septiembre)
        echo "$Mes posee 30 dias"
        ;;
    octubre)
        echo "$Mes posee 31 dias"
        ;;
    noviembre)
        echo "$Mes posee 30 dias"
        ;;
    diciembre)
        echo "$Mes posee 31 dias"
        ;;
    *)
        echo "error"
        ;;
esac
```

3. Modify /etc/profile so that you get a special greeting message when you connect to your system as root.

 # En el bash

sudo gedit /etc/profile

 # Mete en el profile esto

```
if [ $USER == "root" ]; then
        echo "jijiji sos admin"
#opcional
else
        echo "buenas"
fi
```

4. Edit the leaptest.sh script from Section 7.2.4 so that it requires one argument, the year. Test that exactly one argument is supplied.

5. Write a script called whichdaemon.sh that checks if the httpd and init daemons are running on your system. If an httpd is running, the script should print a message like, "This machine is running a web server." Use ps to check on processes.

6. Write a script that makes a backup of your home directory on a remote machine using scp. The script should report in a log file, for instance ~/log/homebackup.log. If you don't have a second machine to copy the backup to, use scp to test copying it to the localhost. This requires SSH keys between the two hosts, or else you have to supply a password. The creation of SSH keys is explained in man ssh-keygen.
7. Adapt the script from the first example in Section 7.3.1 to include the case of exactly 90% disk space usage, and lower than 10% disk space usage.

The script should use tar cf for the creation of the backup and gzip or bzip2 for compressing the .tar file. Put all filenames in variables. Put the name of the remote server and the remote directory in a variable. This will make it easier to re-use the script or to make changes to it in the future.

The script should check for the existence of a compressed archive. If this exists, remove it first in order to prevent output generation.

The script should also check for available diskspace. Keep in mind that at any given moment you could have the data in your home directory, the data in the .tar file and the data in the compressed archive all together on your disk. If there is not enough diskspace, exit with an error message in the log file.

The script should clean up the compressed archive before it exits.

Chapter 8

These exercises are practical applications of the constructs discussed in this chapter. When writing the scripts, you may test by using a test directory that does not contain too much data. Write each step, then test that portion of code, rather than writing everything at once.

1. Write a script that asks for the user's age. If it is equal to or higher than 16, print a message saying that this user is allowed to drink alcohol. If the user's age is below 16, print a message telling the user how many years he or she has to wait before legally being allowed to drink.

As an extra, calculate how much beer an 18+ user has drunk statistically (100 liters/year) and print this information for the user.

```
echo "Ingresa tu edad:"
       read n
       if [ $n -ge 16 ]
       then
       echo "Tenes la edad suficiente para tomar"
       else
       var=`expr 16 - $n`
       echo "Espera $var años para tomar"
       fi
```

```
//////////////////////////////////////////////////////////
#!/bin/bash
echo "Ingrese su edad:"
     read n
     if [ $n -gt 19 ]
     then
     echo "Ja'u kp"
     else
     var=`expr 20 - $n`
     var2=$(( 365*$var ))
     echo "Espera $var2 dias para tomar"
     fi
```

2. Write a script that takes one file as an argument. Use a here document that presents the user with a couple of choices for compressing the file. Possible choices could be gzip, bzip2, compress and zip.

Link:
https://www.networkworld.com/article/3538471/how-to-compress-files-on-linux-5-ways.html

#!/bin/bash

```
# checking if there is at least one filename passed
bigfile=$1
if [ bigfile -ne 1 ]
then
        echo "At least one filename must be passed"
        echo "Usage: $0 "
        exit 1
fi
```

#Compress to gzip, You just type "gzip" followed by the name of the file you want to compress. Unlike the commands described above, gzip will encrypt the files "in place". In other words, the original file will be replaced by the encrypted file.

```
gzip bigfile
ls -l bigfile*
```

#Compress to bzip2, As with the gzip command, bzip2 will compress the file that you select "in place", leaving only the original file.

```
bzip bigfile
ls -l bigfile*
```

#Compress to zip, The zip command creates a compressed file while leaving the original file intact. The syntax is straightforward except that, as with tar, you have to remember that your original file should be the last argument on the command line.

```
zip ./bigfile.zip bigfile
ls -l bigfile bigfile.zip
```

3. Write a script called homebackup that automates tar so the person executing the script always uses the desired options (cvp) and backup destination directory (/var/backups) to make a backup of his or her home directory. Implement the following features:

- **Test for the number of arguments. The script should run without arguments. If any arguments are present, exit after printing a usage message**
- **Determine whether the backups directory has enough free space to hold the backup.**
- **Ask the user whether a full or an incremental backup is wanted. If the user does not have a full backup file yet, print a message that a full backup will be taken. In case of an incremental backup, only do this if the full backup is not older than a week.**
- **Compress the backup using any compression tool. Inform the user that the script is doing this, because it might take some time, during which the user might start worrying if no output appears on the screen.**
- **Print a message informing the user about the size of the compressed backup.**

See info tar or Introduction to Linux, chapter 9: "Preparing your data" for background information.

4. Write a script called simple-useradd.sh that adds a local user to the system. This script should:

- **Take only one argument, or else exit after printing a usage message.**
- **Check /etc/passwd and decide on the first free user ID. Print a message containing this ID.**
- **Create a private group for this user, checking the /etc/group file. Print a message containing the group ID.**
- **Gather information from the operator user: a comment describing this user, choice from a list of shells (test for acceptability, else exit printing a message), expiration date for this account, extra groups of which the new user should be a member.**
- **With the obtained information, add a line to /etc/passwd, /etc/group and /etc/shadow; create the user's home directory (with correct permissions!); add the user to the desired secondary groups.**
- **Set the password for this user to a default known string.**

5. Rewrite the script from Section 7.2.1.4 so that it reads input from the user instead of taking it from the first argument.

Chapter XX

https://github.com/techarkit/shell-scripting-tutorial no se que es pero parece que tiene cosas

Chapter 9

Remember: when building scripts, work in steps and test each step before incorporating it in your script.

1. **Create a script that will take a (recursive) copy of files in /etc so that a beginning system administrator can edit files without fear.**

```
ls /etc > list
for i in `cat list`; do cp /etc/"$i" /etc/copy"$i" ; done
```

2. **Write a script that takes exactly one argument, a directory name. If the number of arguments is more or less than one, print a usage message. If the argument is not a directory, print another message. For the given directory, print the five biggest files and the five files that were most recently modified.**

```
#!/bin/bash
```

```
if [$# -ne 1];
 then echo "illegal number of parameters"
fi

PASSED=$1

if [[ -d $PASSED ]]; then
   echo "$PASSED is a directory"
fi

sudo find / -type f -printf "%s\t%p\n" | sort -n | tail -1
find $PASSED -type f -printf '%s %p\n' | sort -nr | head -n 5
find $PASSED -type f -printf "%T@ %p\n" | sort -nr | head -5
```

3. **Can you explain why it is so important to put the variables in between double quotes in the example from Section 9.4.2?**

4. **Write a script similar to the one in Section 9.5.1, but think of a way of quitting after the user has executed 3 loops.**

```
#!/bin/bash
# This script provides wisdom
# You can now exit in a decent way.
FORTUNE=/usr/games/fortune

i="0"

while [ $i -lt 3 ]
do
echo "On which topic do you want advice?"
echo "1. politics"
echo "2. startrek"
echo "3. kernelnewbies"
echo "4. sports"
echo "5. bofh-excuses"
echo "6. magic"
echo "7. love"
echo "8. literature"
echo "9. drugs"
echo "10. education"
echo
```

```
echo -n "Enter your choice, or 0 for exit: "
read choice
echo

case $choice in
 1)
 $FORTUNE politics
 ;;
 2)
 $FORTUNE startrek
 ;;
 3)
 $FORTUNE kernelnewbies
 ;;
 4)
 echo "Sports are a waste of time, energy and money."
 echo "Go back to your keyboard."
 echo -e "\t\t\t\t -- \"Unhealthy is my middle name\" Soggie."
 ;;
 5)
 $FORTUNE bofh-excuses
 ;;
 6)
 $FORTUNE magic
 ;;
 7)
 $FORTUNE love
 ;;
 8)
 $FORTUNE literature
 ;;
 9)
 $FORTUNE drugs
 ;;
 10)
 $FORTUNE education
 ;;
 0)
 echo "OK, see you!"
 break
 ;;
 *)
 echo "That is not a valid choice, try a number from 0 to 10."
```

```
 ;;
esac
i=$[$i+1]
done
```

5. Think of a better solution than move -b for the script from Section 9.5.3 to prevent overwriting of existing files. For instance, test whether or not a file exists. Don't do unnecessary work!

6. Rewrite the whichdaemon.sh script from Section 7.2.4, so that it:
   - Prints a list of servers to check, such as Apache, the SSH server, the NTP daemon, a name daemon, a power management daemon, and so on.
   - For each choice the user can make, print some sensible information, like the name of the web server, NTP trace information, and so on.
   - Optionally, build in a possibility for users to check other servers than the ones listed. For such cases, check that at least the given process is running.
   - Review the script from Section 9.2.2.4. Note how character input other than q is processed.
   - Rebuild this script so that it prints a message if characters are given as input.

Chapter 10

Here are some brain crackers:
Write a script that does the following:

- Display the name of the script being executed.
- Display the first, third and tenth argument given to the script.
- Display the total number of arguments passed to the script.
- If there were more than three positional parameters, use shift to move all the values 3 places to the left.
- Print all the values of the remaining arguments.
- Print the number of arguments.

Test with zero, one, three and over ten arguments.

```
#!/bin/bash

echo "Nombre del script: $0"
#Se cargan los argumentos en un array
myArray=("$@")
```

```
echo "El array tiene: " ${#myArray[@]} " elementos."
echo ${myArray[*]}

echo ${myArray[1]}, ${myArray[5]}, ${myArray[10]}

//Se eliminan estos argumentos del array
unset myArray[1]
unset myArray[5]
unset myArray[10]

echo "El array ahora tiene: " ${#myArray[@]} " elementos."
echo ${myArray[*]}

//////////////////////
#!/bin/bash

array=("$@")
i=$#
while [ $i -gt -1 ]
do
echo ${array[$i]}
i=$[$i-1]
done
//////////////////////////////////////////////

#!/bin/bash

echo el programa se llama: $0

vector=($@)

echo -e "los argumentos que recibe el script son:\n"${vector[*]}

echo ${vector[0]} ${vector[2]} ${vector[9]}

echo "El array tiene: " ${#vector[@]} " elementos."

if (( $# > 3 ))
then
 i=0

 while (( "$#" )); do
   shift 3
```

```
    if (( "$1" -ne "" )); then
      shifeado[$i]=$1
      i=$[$i+1]
    fi
  done

fi

echo -e "los argumentos que recibe el script son:\n"${shifeado[*]}
echo "El array tiene: " ${#shifeado[@]} " elementos."
```

Write a script that implements a simple web browser (in text mode), using wget and links -dump to display HTML pages to the user. The user has 3 choices: enter a URL, enter b for back and q to quit. The last 10 URLs entered by the user are stored in an array, from which the user can restore the URL by using the back functionality.

Chapter 11

Here are some useful things you can do using functions:

1. **Add a function to your ~/.bashrc config file that automates the printing of man pages. The result should be that you type something like printman <command>, upon which the first appropriate man page rolls out of your printer. Check using a pseudo printer device for testing purposes.**
   As an extra, build in a possibility for the user to supply the section number of the man page he or she wants to print.

   ```
   #agregar en ~/.bashrc
   printman()
   {
    man "$1"
   }
   ```

2. **Create a subdirectory in your home directory in which you can store function definitions. Put a couple of functions in that directory. Useful functions might be, amongs others, that you have the same commands as on DOS or a commercial UNIX when working with Linux, or vice versa. These functions should then be imported in your shell environment when ~/.bashrc is read.**

   ```
   #import functions defined by user en la carpeta especificada
   for i in $HOME/functions/*;
   ```

```
 do source $i
done
```

Chapter 12

A couple of practical examples:

1. **Create a script that writes a boot image to a diskette using the dd utility. If the user tries to interrupt the script using Ctrl+C, display a message that this action will make the diskette unusable.**

   #! /bin/bash
   #Escribir imagen del disco de arranque de Linux en un disquete.
   #Inserte un disco pero no lo monte

   trap "echo No lo haga compa, su disquete se va a fundir" SIGINT
   dd if=disk.img of=/dev/fd0 bs=1440k

2. **Write a script that automates the installation of a third-party package of your choice. The package must be downloaded from the Internet. It must be decompressed, unarchived and compiled if these actions are appropriate. Only the actual installation of the package should be uninterruptable.**

http://marqueta.org/code/2013/11/21/traps-y-senales-en-bash-i.html

```
cat ctrlc.sh
#!/bin/bash
trap 'echo "Paso de ti..."' SIGINT
while true; do
 sleep 1
done
```

****

si tenemos un script que genera ficheros temporales que queremos eliminar si termina de forma inesperada, podemos redefinir ciertas señales de este modo:

```
#!/bin/bash

#

# Borrar fichero temporal:
```

```
trap '/bin/rm /tmp/mitemp.$$; exit' SIGHUP SIGINT SIGQUIT SIGTERM
```

****

```
#!/bin/bash -e
# Defino el trap
trap 'echo "Se ha producido un error"' ERR
touch testfile.txt
chmod 444 testfile.txt
echo "esto va a fallar" >testfile.txt
echo "esto no se va a ejecutar"
```

hasta la proximaaaa