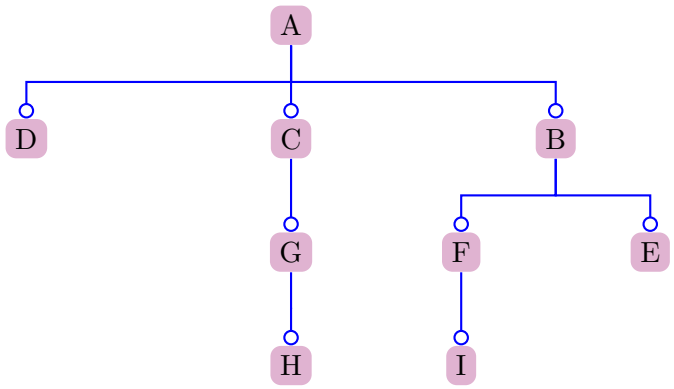
Homework 1

Exercice 1

Text

Build the following directory tree:



modify permissions of directories in order that:

1. everybody can read and write in a;
2. nobody can create subdirectories of d;
3. others can not see inside c, but can create brothers of g;
4. others can enter in g, but not in h;
5. group can not see g (visualize what is inside g), but can enter in h;
6. others has no rights on b, while group has all permissions;
7. owner can read i, but has no other rights on i.

Create a file *xyz* in a (using touch command) and modify permissions in order that only the owner can read and write on the file.

Solution

Directories are create with the following commands:

```
mkdir a
cd a
mkdir b
mkdir c
mkdir d
cd c
mkdir g
```

```
cd g
mkdir h
cd..
cd ..
cd b
mkdir e
mkdir f
cd f
mkdir i
```

Permissions - directory

1. `chmod 777 a`;
2. `chmod 555 d`;
3. `chmod 772 c`;
4. `chmod 770 h`;
5. `chmod 717 g`;
6. `chmod 770 b`;
7. `chmod 477 i`.

```
cd a
touch xyz        - create the file
ls               - check if file exist
chmod 600 xyz
ls -la           - check permission
```

Exercice 2

Text

Verify with which permissions files and directories are usually created. Then, using umask command, try to have these files created readable and writable from everyone. Try to have these files readable and writable only from the user.

Solution

Fist of all a file need to be created:

```
touch bbb
```

With the command `ls -al` is possible to verify default permissions; they are:

```
-rw-r--r--  1 user user        0 2010-11-30 15:32 bbb
```

If we want that every file will be readable and writable to everyone we have to allow that permission:

```
umask 001
```

then we create another file:

```
touch wsv
ls -al
```

The resultant permission will be:

```
-rw-rw-rw-  1 user user     0 2010-11-30 15:39 wsv
```

Otherwise if we want to have files readable and writable only for user the command list will be:

```
umask 177
touch wev
ls -al
```

The resultant permission will be:

```
-rw-------  1 user user     0 2010-11-30 15:46 wev
```

Exercice 3

Text

1. Find all files in the system named core;
2. Find all the files whose name contains the string conf and show its size;
3. Find all files of the bin user, put in the /usr directory (and in its subdirectories), whose permissions are *–r-xr-xr-x*;
4. Find all files of yours that have been seen from less than a week.

Solution

1. `find / -name core`;
2. `find / -name '*conf*'-printf '%p: %s bytes\n'`;
3. `find /usr -user bin -perm 555`;
4. `find / -atime -7 -user $USER`.

Homework 2

Exercice 1

Text

Show a list of all files called "core" in all the file system. Do not visualize error messages.

Solution

This work can be done easly with the following command:

```
find /  -name '*core*' -print 2> error.txt
```

The most important thing to remember is that `2>` send all errors retreived by the stderr to the file *error.txt*.

In order to check if the command worked it is possible to open that file with any kind of text editor; using gedit:

```
gedit error.txt
```

This command shows how many errors are present.

Exercice 2

Text

Show a list of all processes of root user. The list must have the following format:

PID	%CPU	COMMAND

Solution

There are at least two solutions for this problem. The first one is:

```
ps -u root -eo pid -eo pcpu -eo com
```

The second one is a bit more difficult since it is required to parse the entire content in this way:

```
ps -el | sed 's/\s\+/\t/g' | cut -f 4,6,14
```

Exercice 3

Text

Create 15 empty files called [xxx1, xxx2, xxx3 , ... , xxx15]. Show the list of the previously created file names both in ascending and descending order. Save the list in a file called *list.txt*.

Solution

First of all files have to be created in a directory:

```
mkdir test
cd test
touch file1
touch file2
touch file3
touch file4
touch file5
touch file6
touch file7
touch file8
touch file9
touch file10
touch file11
touch file12
touch file13
touch file14
touch file15
```

In order to do this, it is also possible create a script, but since now it is not necessary: the previous way it is more simple and effective.

The second step is order them in an ascending order and save that list into a file; it is required only a command:

```
ls | sort -n > list_ascending_order.txt
```

Instead, to have a list of them in a descending order is sufficient change the option `-n` into `-r`:

```
ls | sort -r > list_descending_order.txt
```

Exercice 4

Text

Find all the directories of your home whose name ends with the string *backup* (eventually create some of them) and move them in a backup directory.

Solution

This problem can be solved with the following command:

```
find ~ -name '*backup' i- exec mv \{} ./ locks \; -type d
```

The character ~ specify quicky the directory home.

Exercice 5

Text

Find all files in the system that are greater than 2 MByte or whose name begins with *a* and finishes with *o*.

Solution

This exercise is solved using:

```
find / \( -size +2M -o -name 'a*o' \ )
```

To specify all names whose start letter is *a* and end letter is *o* it is used the option `-name 'a*o'`.

Homework 3

Exercice 1

Text

Read from stdin a set of strings, stopping when the string is *END*. Copy and number each line on stdout.

Solution

In order to solve this problem we create the following script:

```
#!/bin/bash
read str
i=0
while [ $str != 'END' ];
do
echo $i $str
let i=i+1
read str
done
```

Below is reported an example showing how the script works:

```
user@user-desktop:~/Scrivania$ ./reading_strings.sh
hello
0 hello
this
1 this
is
2 is
an
3 an
example
4 example
END
user@user-desktop:~/Scrivania$
```

Exercice 2

Text

Given two numbers as parameters from the command line, visualize a rectangle with dimensions given from the two parameters.

Solution

The following script solve the problem:

```
#!/bin/bash
x1=$1
x2=$2
i=0
let fin2=x2-2
let fin1=x1-2
let w=x2-1
echo -n +
while [ $i -lt $fin2 ]
    do
        echo -n -
        let i=i+1
done
echo +
for (( j=1 ; j<=fin1 ; j++ ))
        do
        for (( i=0 ; i<=w ; i++))
        do
                if [ "$i" -eq 0 ] || [ "$i" -eq $w ]
                        then
                        echo -n \|
                else
                        echo -n " "
                fi
        done
```

```
        echo
done
i=0
echo -n +
while [ $i -lt $fin2 ]
    do
        echo -n -
        let i=i+1
done
echo +
```

Using the script is possible to check the outcome:

```
user@user-desktop:~/Scrivania$ ./create_rectangle.sh 5 10
+--------+
|        |
|        |
|        |
+--------+
user@user-desktop:~/Scrivania$
```

Exercice 3

Text

Verify that the PATH variable contains the /usr/local/bin directory and print a message with the result of the verification.

Solution

This exercise is very simple: it is just request to check if a string contains another string. The following script is able to do this job:

```
#!/bin/bash

sequenza_nota=$PATH
sequenza_lettere=/usr/local/bin
if echo "$sequenza_nota" | grep -q "$sequenza_lettere"
then
  echo "\"$sequenza_lettere\" found!"
else
  echo "\"$sequenza_lettere\" not found!"
fi
```

Notice that the option `-q` is able to cancel the output that `echo` creates.

Exercice 4

Text

Write a script able to read two parameters from the command line. The two parameters represent the names of two directories. Copy all the files from the first directory to the second, modifying each file in order to substitute all strings *DAY* with the string *GIORNO*.

Solution

To solve this problem we have tought to follow these steps:

. first copy all files from the first directory into a temporary directory: this avoids modifications on those files;

. with sed all changes are done;

. at the end all files in the temporary directory are removed, then the directory is cancelled (notice that to destroy a directory, mandatory the directory must be empty).

The script used is:

```
#!/bin/bash
mkdir temp
cd $1
cp * ../temp
cd ../temp
for i in *
do
sed -i 's/DAY/GIORNO/g' * | cp * ../$2/
done
rm *
cd ..
rmdir temp
```

Actually, the condition in which the `for` works can be realized in two ways:

. the one adopted by the script reported;

. using: `for i in temp`.

Both solutions are valid and work in the same way; perhaps the second one is more clear than using the *jolly* character *.

Exercice 5

Text

Given two parameters from the command line, the first as a directory name and the second as the size in bytes of a file, visualize all ordinary files in the

specified directory that can be read (read permission enabled) and with a size greater than the specified one. Please also verify that the given parameters are correct.

Solution

This exercise is a bit more difficult and it is used to show how functions work in bash. The script is:

```
#!/bin/bash

check_num_par(){
  # this function prints, if there are, parameters received
  if [ $# != 0 ]
  then
     echo "parameters are: $* "
  else
     echo "no parameters are inserted!"
fi

}

dir_name=$1
file_size=$2
check_num_par $dir_name $file_size
if [ -d $dir_name ]
then
  cd $dir_name
  find . \(-size +"$file_size" \(-perm 444 \) \) 2> /dev/null
else
  echo "$dir_name is not a directory!"
fi
```

Let us start introducing functions. In bash, functions can be created in two different ways:

. first mode:

```
function name_function {
commands...
}
```

. second mode:

```
name_function() {
commands...
}
```

To call them is simply required to use the name; pay attention that, if internally there are conditions that check the value of some parameter, those parameters have to be passed to the function. Actually this is exactly the

case in which we are interested: with the function `check_num_par` we want to check if parameters are present so the call will be: `check_num_par $par1 $par2`. Notice that the function considered is able to check and print an infinite number of parameters, not only two. If using the script no parameters are typed, the function report this error on the screen.

What concern the exercise, first of all parameters are assigned at two variables, then using the function mentioned previously, a first check is performed, to be sure that they are not null. Since one constraint of the exercise is verify that parameters are correct a second check is performed to control if the directory name is really a directory; this is done by `if [ -d $dir_name ]`.