Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadores (DISCA)
Universitat Politècnica de València

Part 1: Introduction

Seminar 2

The UNIX shell

fSO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Introduction to UNIX

- **Objectives**:
  - Using the UNIX shell and understanding its operation
  - Knowing the basic structure of the file system in UNIX
  - Understanding and using the concept of file path
  - Knowing and using some useful shell variables and commands

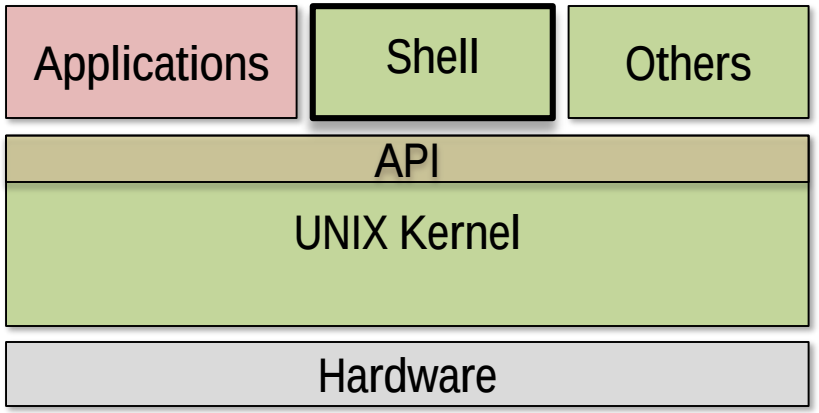
- **Bibliography**
  * Kernighan, Pike: The UNIX programming environment/El entorno de programación UNIX, Prentice Hall
  * Fernando Bellas: El entorno de programación UNIX http://www.tic.udc.es/~fbellas/teaching/unix/index.html
  - Mark Burgess: The UNIX programming environment http://www.iu.hio.no/~mark/unix/unix_toc.html

(*) Available in PoliformaT (Recursos -> Complementary material -> Docs)

Content

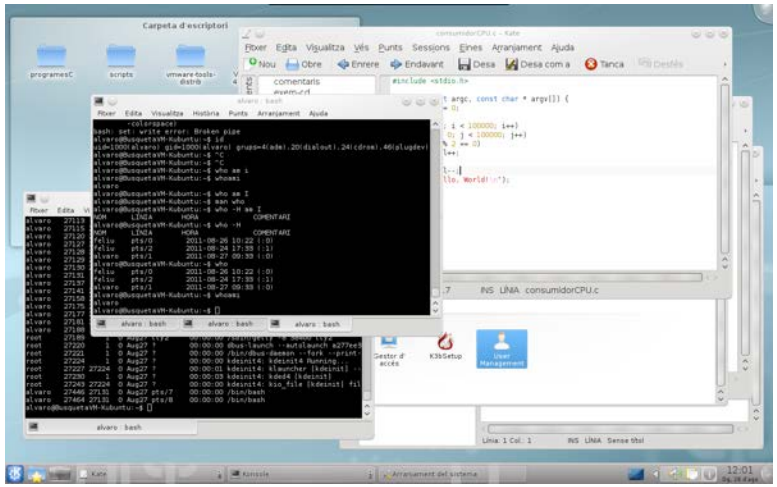
- **Introduction**
- Users and groups
- Shell variables
- Browsing the file system
- File system management
- Input / output management
- Process management
- Shell programming

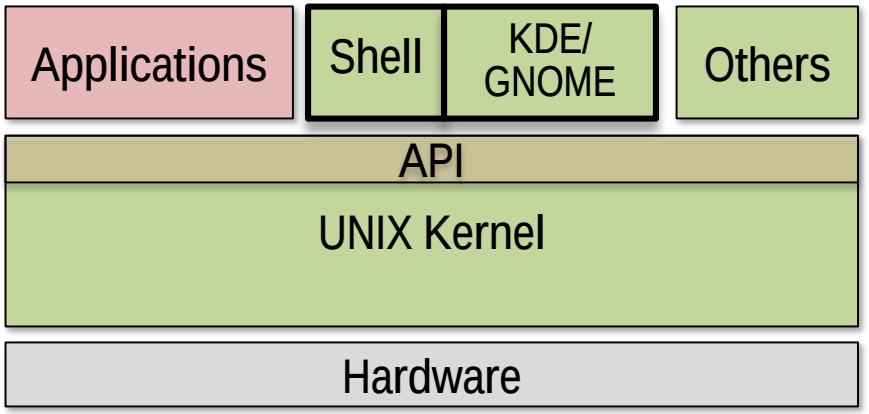
- The user interfaces for UNIX
  - The user interfaces are necessary to implement programs, to organize files, to perform system administration, etc
  - In any case, a user interface (graphical or text) is a program that translates user actions into calls to system services to perform the required task.
  - The classical text interface or console is made of a keyboard and a monitor in text mode
    - Inside UNIX: device tty0, tty1, ...
  - Command language: UNIX shell



Introduction

- UNIX graphical user interfaces (GUI)
  - *X Window* (1984): *X11* (or simply *X)*
    - Implementations:
      - *XFree86*, *X.Org* (>= 2004)
      - *X Quartz*
      - *Cygwin/X*, *mobaXterm*, *Xming*, … *FreeNX*
      - *Android X Server*
    - *Client/Server architecture -> remote conection*
    - *Several simultaneous terminal windows (xterm) can be used*
    - It is designed for graphical monitors using keyboard and mouse
  - Moders Desktops (Unity, KDE, GNOME, etc) provide GUI tools to use and to administrate the system





Terms

KDE: Powerful graphical desktop environment for Unix workstations
GNOME: GNU Object Model Environment

- **UNIX shell**
  - It is a program that interprets commands (text lines) and runs them
  - The syntax supported by the shell allows specifing shell scripts as programs
    - It has comments, variables, flow control, functions (with arguments and return values, etc.)
    - The shell can process text files including command programs
  - There are several shells (similar but different): cshell, kshell ...
    - We are going to use **bash or Bourne Again Shell**
  - The UNIX shell is able to:
    - Manage the file system
    - Run programs
    - Control programs input / output
    - Maintain system variables
    - Monitor and manage processes

- Shell behaviour

  Repeat

    write the *prompt*

    read the line entered by the user

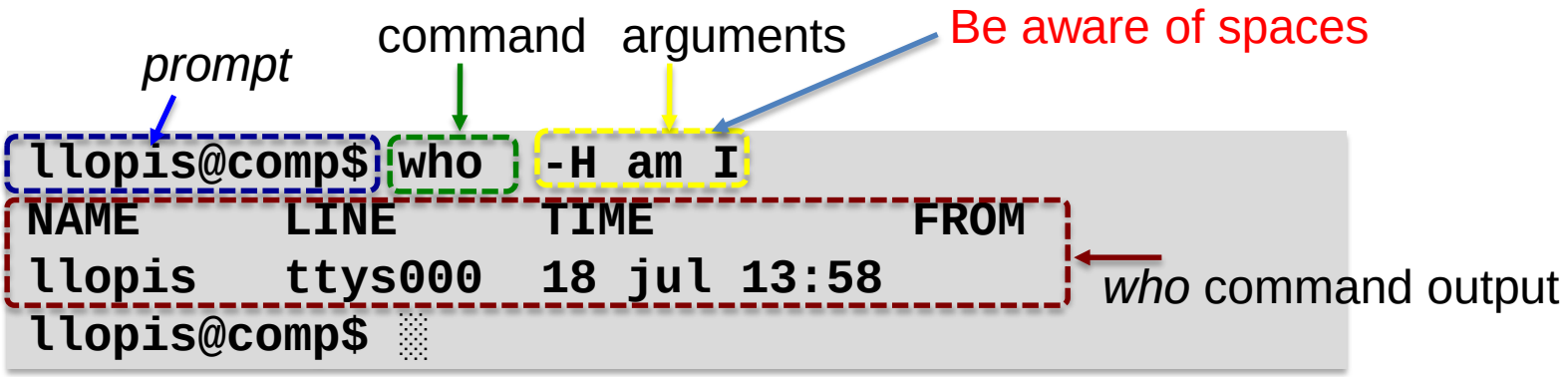
    process the line (expanding metacharacters)

    identify the commad (the first line token or word)

    execute the command passing the arguments (all other line tokens)

    receive the program termination code

  until (end of the input file)



Metacharacters: **$ * ? \ > < | & [ ]**

- **Internal and external commands**
  - The shell deals with two type of commands:
    - Internal or built-in commands:
      - The shell includes internal code to perform simple and frequent operations without relying in other programs
    - External commands: the shell seeks a program file and runs it
      - Binary executable: like a C program compiled and linked
      - Script: text files that an interpreter (bash, Python, Perl, etc) executes line by line
  - **Online help: man command**
    - Description of a specific command use: `$ man command_name`
    - Description of the shell: `$ man bash`

```
LS(1)                    BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLPRTWZabcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory,
```

Content

- Introduction
- **Users and groups**
- Shell variables
- Browsing the file system
- File system management
- Input / output management
- Process management
- Shell programming

Users and groups

- They are the base of the UNIX protection scheme
  - Their management is a very important administrative task in UNIX
- A **user** is an identity known by the system that has a password, a privilege set, a home directory, etc.
  - Every user has a name (username) and a number (**UID**)
  - There are system users like daemon, mail and others
  - The predefined user *root* has the highest privilege level
- A **group** is a set of users
  - Every group has a name and a number (**GID**)
  - There are predefined groups: adm, man, etc.
  - Every user belongs necessarily to a primary and optionally to secondary groups as required

Terms:
UID: User Identification
GID : Group Identification

Users and groups

Command	Operation
id	Shows the user identity and the groups that the user belongs to
su	Changes the active user
who	Gives a list of the actual active users

```
…$ id
uid=1001(feliu) gid=1001(fso)
grups=4(adm),20(dialout),24(cdrom),…
…$ su llopis
Password:
```

Content

- Introduction
- Users and groups
- **Shell variables**
- Browsing the file system
- File system management
- Input / output management
- Process management
- Shell programming

Shell variables

- What are shell variables?
  - They are symbols that have assigned a string value
    - A symbol in the form `$shell_var_name` is replaced by the shell variable value
  - To define a shell variable and assigning it a value:

    `shell_var_name=`*value*

```
…$ ElMeuNom=Feliu
…$ echo ElMeuNom
ElMeuNom
…$ echo $ElMeuNom
Feliu
…$
```

- The environment
  - It is the set of defined shell variables (with `export`) on the shell session
  - The `env` command lists the value of all environment variables

```
…$ env
TERM=xterm
SHELL=/bin/bash
USER=gandreu
```

Shell variables

- Commands to manage shell variables

Command	Purpose
=	Creates a variable and assigns a value to it
export	Sets a variable as environment variable
env	List the values of all the shell variables
echo	Shows the string argument on the standard output

- Some shell variables

Shell variable	Purpose	Show value command
PATH	Contains the location path of executable files	**echo $PATH**
HOME	Path of the user home directory	**echo $HOME**
HOSTNAME	Name on the computer on the network	**echo $HOSTNAME**
PS1	*Prompt* look	**echo $PS1**
PWD	Actual working directory	**echo $PWD**
SHELL	Default command interpreter	**echo $SHELL**
TERM	Terminal type	**echo $TERM**

Shell variables

- **The shell variable PATH**
  - It contains the list of directories where the shell looks for programs and scripts when processing the command line:
    - Directory names are separated by the character ‘ : ’
    - When looking for programs, the shell follows the directory order

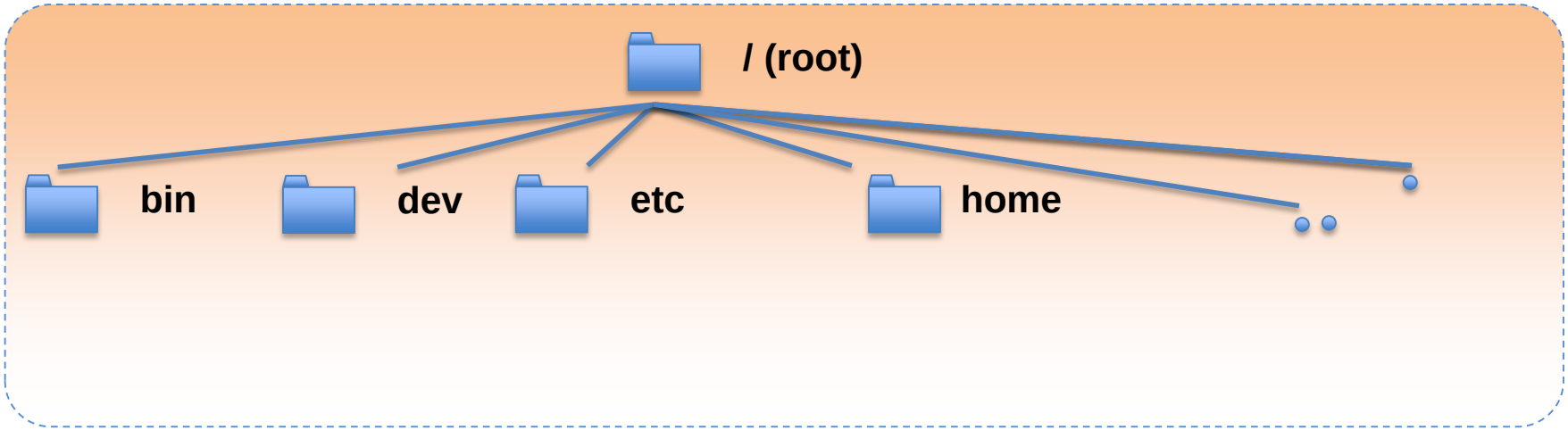
```
…$ echo $PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
…$ export PATH=$HOME/proves:$PATH
…$ echo $PATH
/Users/llopis/proves:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
…$
```

Content

- Introduction
- Users and groups
- Shell variables
- **Browsing the file system**
- File system management
- Input / output management
- Process management
- Shell programming

Browsing the file system

- **Common UNIX File system scheme**



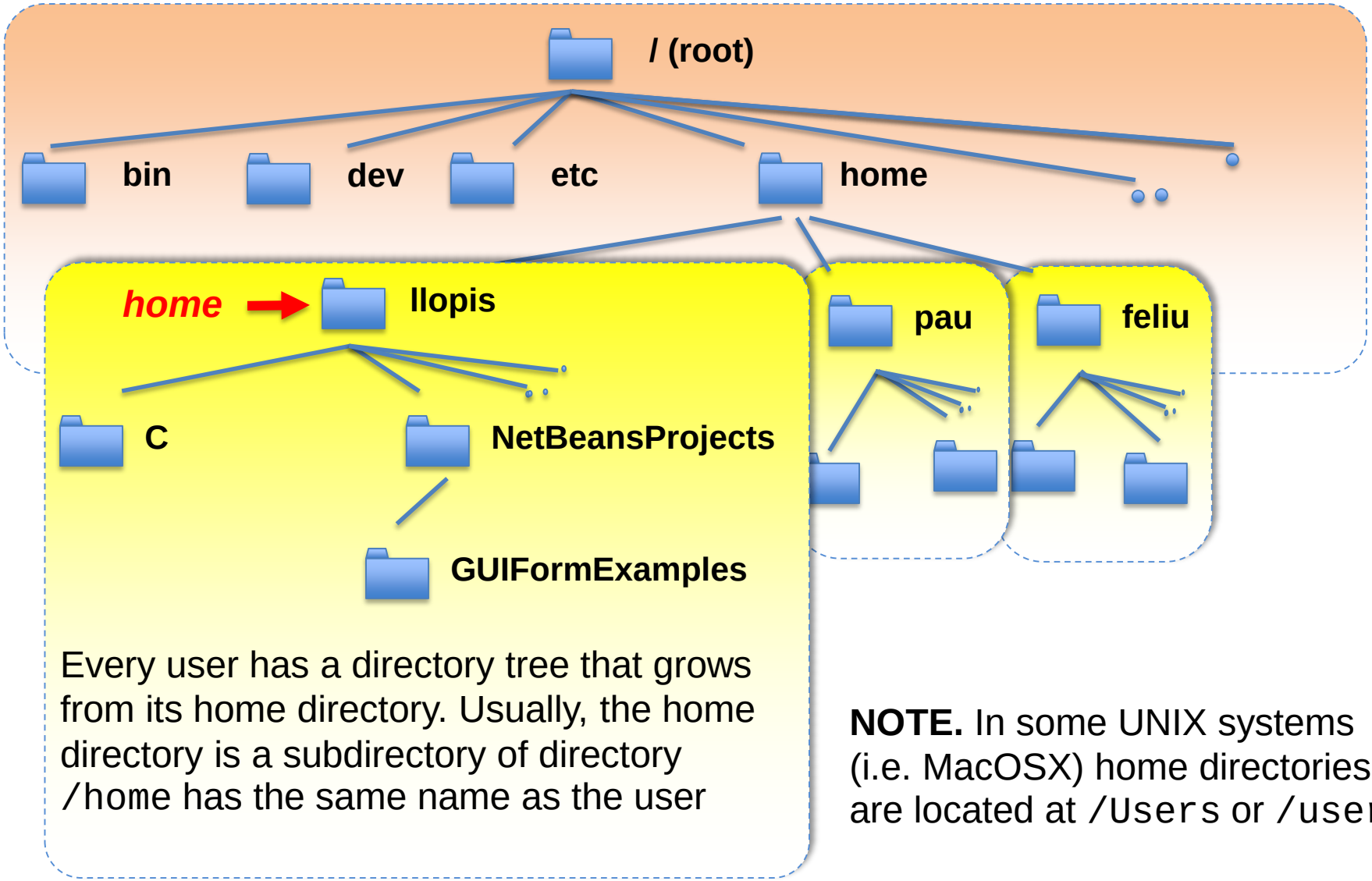
- General use directories
  - They contain the programs and data useful for the system and users
- Directory "."
  - Refers to the actual working directory
- Directory ".."
  - Refers to the parent directory

Examples

Directory	Kind of conteined files
/bin	Common commands
/dev	I/O devices
/etc	Administration files
/usr	Application files

Browsing the file system

- Common UNIX File system scheme (2)



Every user has a directory tree that grows from its home directory. Usually, the home directory is a subdirectory of directory `/home` has the same name as the user

NOTE. In some UNIX systems (i.e. MacOSX) home directories are located at `/Users` or `/user`

Browsing the file system

- **File and directory names in UNIX**
  - Directories form a tree (inverted) growing from the root
  - The full path name of a file or directory contains the names of all the directories above it
    - The name separator is the character ‘ / ’
    - The root directory is ‘ / ’
  - The **absolute path** starts from the file system root directory
    - Example: “/users/llopis/eclipse/p1/main.java”
  - The **relative path** starts from a base directory (let’s consider “/users/llopis/eclipse”)
    - Example: “p1/main.java” is the same as the above absolute path
    - Self directory ” . " refers to the same base directory
      - “.” is the same base directory “/users/llopis/eclipse”
    - Parent directory “ .. " is the directory that contains the base directory
      - “..” is the directory “/users/llopis”

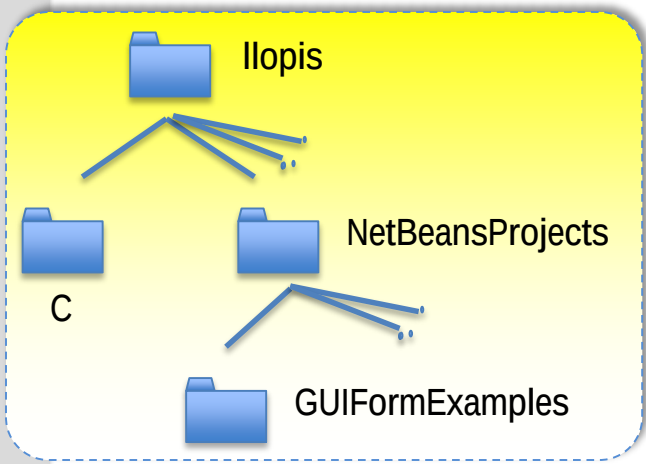
Browsing the file system

- The working directory

Command	Arguments	Operation
cd	*directory*	Establishes the working directory
pwd		Shows the actual working directory

  - `$ cd [dir]` changes the working directory
    - Without arguments set the home directory as the working directory
  - `$ pwd` displays the absolute path of the actual working directory
  - When a shell starts the working directory is the user home directory

```
…$ pwd
/Users/llopis
…$ cd NetBeansProjects
…$ pwd
/Users/llopis/NetBeansProjects
…$ cd GUIFormExamples
…$ pwd
/Users/llopis/NetBeansProjects/GUIFormExamples
…$ cd
…$ pwd
/Users/llopis
…$
```



Content

- Introduction
- Users and groups
- Shell variables
- Browsing the file system
- **File system management**
- Input / output management
- Process management
- Shell programming

File system management

Command	Options	Arguments	Operation
cat		*file(s)*	Shows the file(s) content
ls	**-l -a -d**	*file(s)*	Lists file(s) attributes
file		*file(s)*	Gives the file(s) type
rm	**-R**	*file(s)/dir(s)*	Removes links, files or directories
mkdir		*directory*	Creates a directory
rmdir		*directory*	Removes an empty directory
mv		*File file* *file(s) dir*	Renames/reallocates files or directories
cp	**-r**	*file file* *file(s) dir*	Copies files or directories
ln	**-s**	*file file* *file dir*	Creates new links to files or directories
chown	**-R**	*user file(s)/dir(s)*	Sets owner
chmod	**-R**	*mode file(s)/dir(s)*	Sets premissions

File system management

- The `cat` command
  - Print on the screen the contents of a file (either text file or not)
  - Not applicable to directories
- The `ls` command
  - Lists names and other file attributes:

    Use: `$ ls [options] [names...]`
    - options: indicate what file info to show

      `-l` (from long): permissions, owner, group, ... (every thing)

      `-d` provides directory information instead of files
    - names: directory or file

      if the name is a directory, it lists the file links

      without names, it lists the working directory
  - Special treatment of hidden files/directories
    - By default: names that start with dot (.) do not appear in the list
    - **`-a`** shows all names

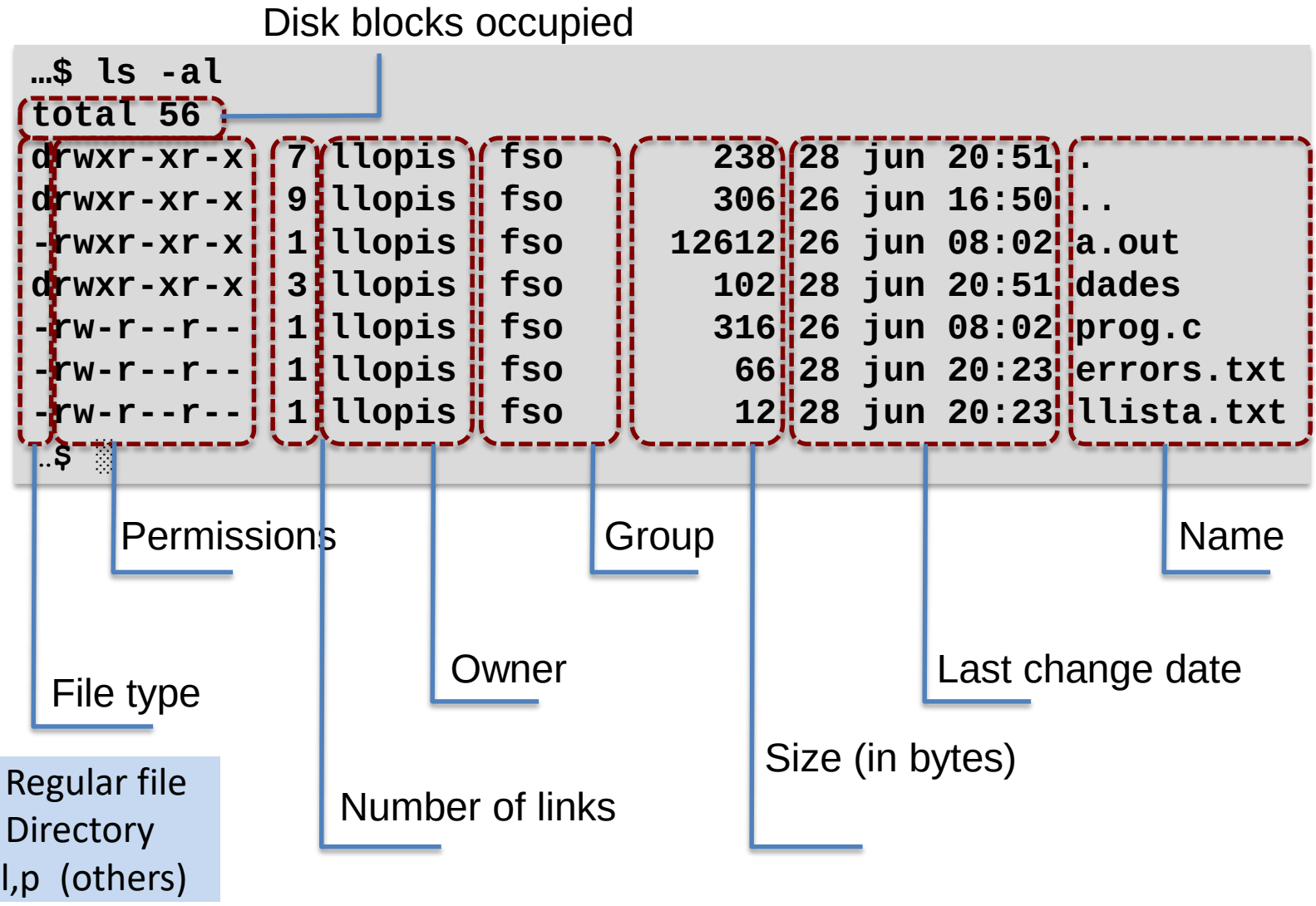
File system management

- File name patterns
  - When a token contains a wildcard, the shell does pattern matching with the entries in a directory and it replaces the initital token by all the names that fit the pattern
  - Wildcard examples:
    - **'*'** matches any substring
    - **'?'** matches any character

```
…$ ls
salida.txt ejemplo.txt param param.c Param algo.
…$ ls [Pp]*
Param param param.c
…$ ls *.*
algo. salida.txt ejemplo.txt param.
…$ ls *.txt
salida.txt ejemplo.txt
…$ ls p*
param param.c
```

File system management

- The format of the long form of the `ls` command



- **File owners and permissions**
  - A file is owned by a user and a group
    - Initially, the owners are the user who created the file and the primary group to which the user belongs
    - The command `chown` (change owner) allows to change the owners
  - Three separate permissions: r (readable), w (writable), x (executable)

Permission	Common files	Directories
r	file reading allowed	the directory can be listed with **`ls`** (only names)
w	file writing allowed	files and directories can be added and deleted to the directory (**x** permission is required)
x	file (binary or script) execution allowed	(together with r permission) files and subdirectories in the directory can be accessed according to permissions set on them

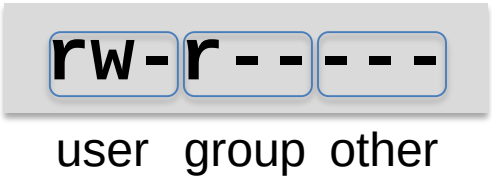
- Three permission domains: user, group and other
- How user U permission apply to a directory D:

if (U is root) permission is given
else if (U is the owner of X) *user* permissions apply
else if (U is the group owner of X) *group* permissions apply
else *other* permissions apply

File system management

- Permission management with *chmod* command
  - There are 9 bits: 3 domains (user, group, other) x 3 permision types

  - Octal representation:
    - Three digits. Every digit encodes permissions for a domain with values from 0 to 7
    - Example: `$ chmod 640 name`



  - Symbolic mode
    - Modifies permissions one by one
    - Examples:

```
$ chmod u+w nom
$ chmod +x nom
$ chmod a+r,o-w nom
$ chmod ug=rw,o= nom
```

Domain	Operation	Permission
u (user) g (group) o (other) a (all)	+ (add) – (remove) = (set)	r w x

Content

- Introduction
- Users and groups
- Shell variables
- Browsing the file system
- File system management
- **Input /output management**
- Process management
- Shell programming

Input /output management

- Standard devices
  - The shell handles three characters channels, every one can be assigned to a device like the console or a file
    - The standard input *stdin (id 0)*
    - The standard output *stdout (id 1)*
    - The standard error output *stderr (id 2)*
  - By default, in an interactive session, all three channels are assigned to the console
  - The shell writes to stdout and reads the command line from stdin
  - Text I/O programs can read from stdin and write to stdout or stderr
  - Example (`ls`):

```
…$ ls
a.out       programa.c
…$ ls q
ls: q: No such file or directory
…$ ls p a*
ls: p: No such file or directory
a.out
…$
```

Standard output

Error output

- **Redirection**
  - Allocation of standard input and outputs to a specific device (typically a file)
  - Command line redirection syntax:

Syntax	Operation
< device	redirects *stdin* to the device
> device	redirects *stdout* to the device (overwrites previous content)
>> device	redirects *stdout* to the device (appends to the end)
2> device	redirects *stderr* to the device (overwrites previous content)
2>&1	redirects *stderr* to the device associated to *stdout*

```
…$ ls p a*
ls: p: No such file or directory
a.out
…$ ls q a* > llista.txt 2> errors.txt
…$ cat llista.txt
a.out
…$ cat errors.txt
ls: q: No such file or directory
…$
```

stdout redirected to *llista.txt*

stderr redirected to *errors.txt*

- **Filters**
  - Commands that read from *stdin*, perform simple operations and write the result to *stdout*

Command	Options	Argument	Operation
head	**-n lines**		Writes the *n* firtst read lines (only one digit)
tail	**-n lines**		Writes the *n* last read lines
sort			Sorts the text lines read and writes the result
tee		*file*	Writes the input in the output and makes a copy in a file
wc	**-l -w -c**		Counts lines, words and characters read and writes the statistics
grep		*expresssion*	Writes the lines that conform to a regular expression
awk			Does pattern matching processing on file content
cut	**-f –d**	*regexp*	selects componets from the text line processed
sed		*script file*	character stream editor

- As all shell commands these ones can be chained: in sequence with ';' or in connection with '|'
- *"regexp" (regular expressions*): describe a pattern to match strings using metacharacters like \ ^ $. [ ] { } | ( ) * + ?

- Filters (examples)

```
…$ cat entrada
one
two
three
four
five
…$
```

```
…$ head -3 <entrada
one
two
three
…$ tail -4 <entrada
two
three
four
five
…$ wc <entrada
       5       5      24
…$
```

```
…$ grep fi <entrada
five
…$ grep t <entrada
two
three
…$ sort <entrada
five
four
one
three
two
…$
```

```
…$ head –n 3 < entrada; grep fi < entrada
…$ grep fi <entrada; echo “Hay`wc –l entrada` coincidencia/s”
```

Input /output management

- **Pipes**
  - They connect the stdout of one command to the stdin of the following command

```
…$ sort <entrada | head -3
five
four
one
…$ sort <entrada | tail -3 >eixida
…$ cat eixida
one
three
two
```

```
…$ ls -l
total 80
-rwxr-xr-x  1 feliu   fso     12612  3 jul 08:41 a.out
drwxr-xr-x  3 feliu   fso       102 13 ago 22:34 e-s
-rw-r--r--  1 feliu   fso         0  3 jul 08:24 eixida.txt
-rw-r--r--  1 feliu   fso        37  3 jul 08:24 exemple.txt
-rwxr-xr-x  1 feliu   fso     12612  3 jul 08:47 param
…$ ls -l | tail -5 | head -3
-rwxr-xr-x  1 feliu   fso     12612  3 jul 08:41 a.out
drwxr-xr-x  3 feliu   fso       102 13 ago 22:34 e-s
-rw-r--r--  1 feliu   fso         0  3 jul 08:24 eixida.txt
```

```
…$ df | sort –rnH|head -1
…$ ps –ax | grep firefox | cut –f 1 –d “ “
```

Content

- Introduction
- Users and groups
- Shell variables
- Browsing the file system
- File system management
- Input /output management
- **Process management**
- Shell programming

Process management

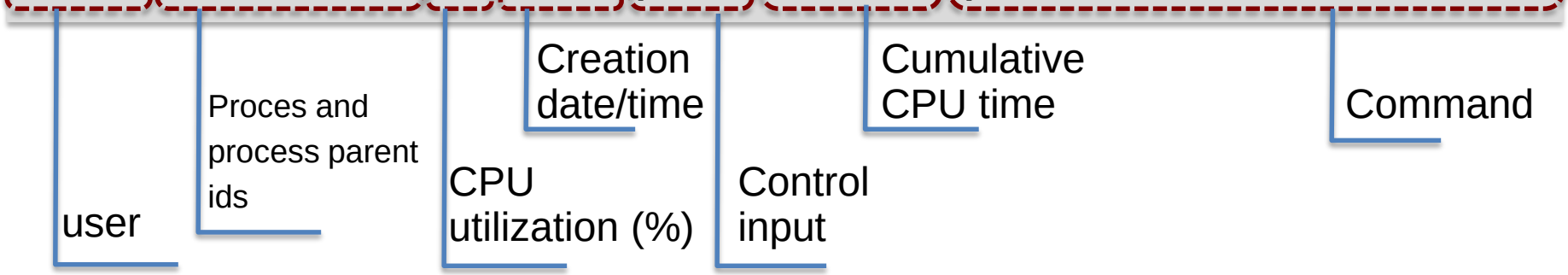
- **Processes in UNIX**
  - A process is identified by its **PID (process identifier)**
    - The shell shows its own PID with `echo $$`
  - Each process is associated with a user with its UID
  - The active process set has a tree structure
    - Every one has a **parent process** defined by its PPID (parent process identifier)
    - Every process can create child processess
- **Commands:**

Command	Usefull options	Arguments	Operation
`ps`	**`-e -f a f`**	*pid(s)*	Lists current processes info
`kill`	**`-s -n`**	*signal pid(s)*	Sends a signal to processes with pid(s)
`sleep`		*time*	Suspends the shell execution during "time"
`pstree`			Shows the process tree
`top` **`htop`**			Shows real time statistics about current active processes

Process management

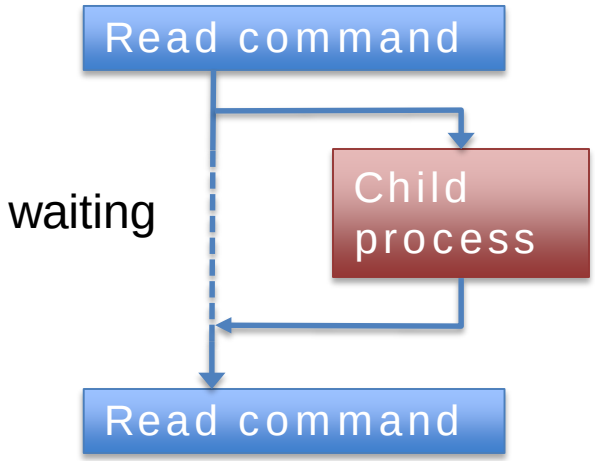
- The command `ps -ef`

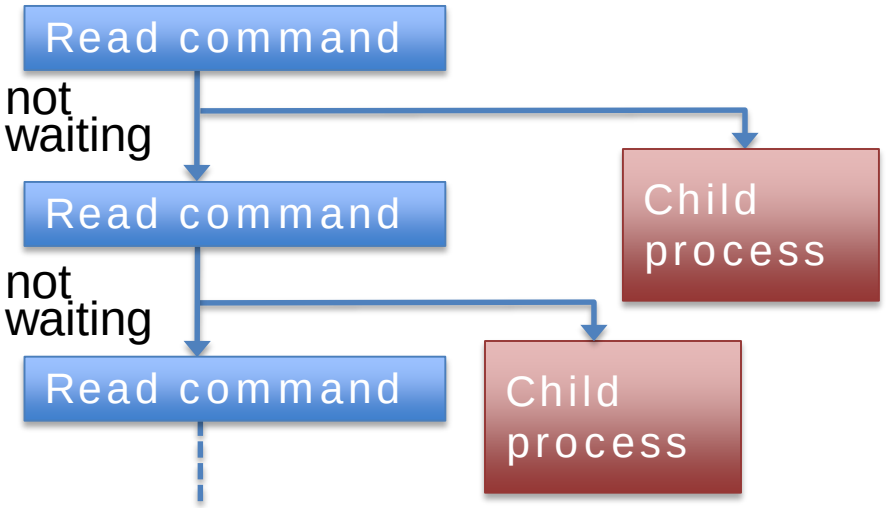
```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Aug24 ?        00:00:02 /sbin/init
root         2     0  0 Aug24 ?        00:00:00 [kthreadd]
…
feliu    19892     1  6 10:23 ?        00:00:00 kdeinit4:konsole [kdeinit]
feliu    19894 19892  9 10:23 pts/1    00:00:00 /bin/bash
feliu    19914 19894  0 10:23 pts/1    00:00:00 ps -ef
```



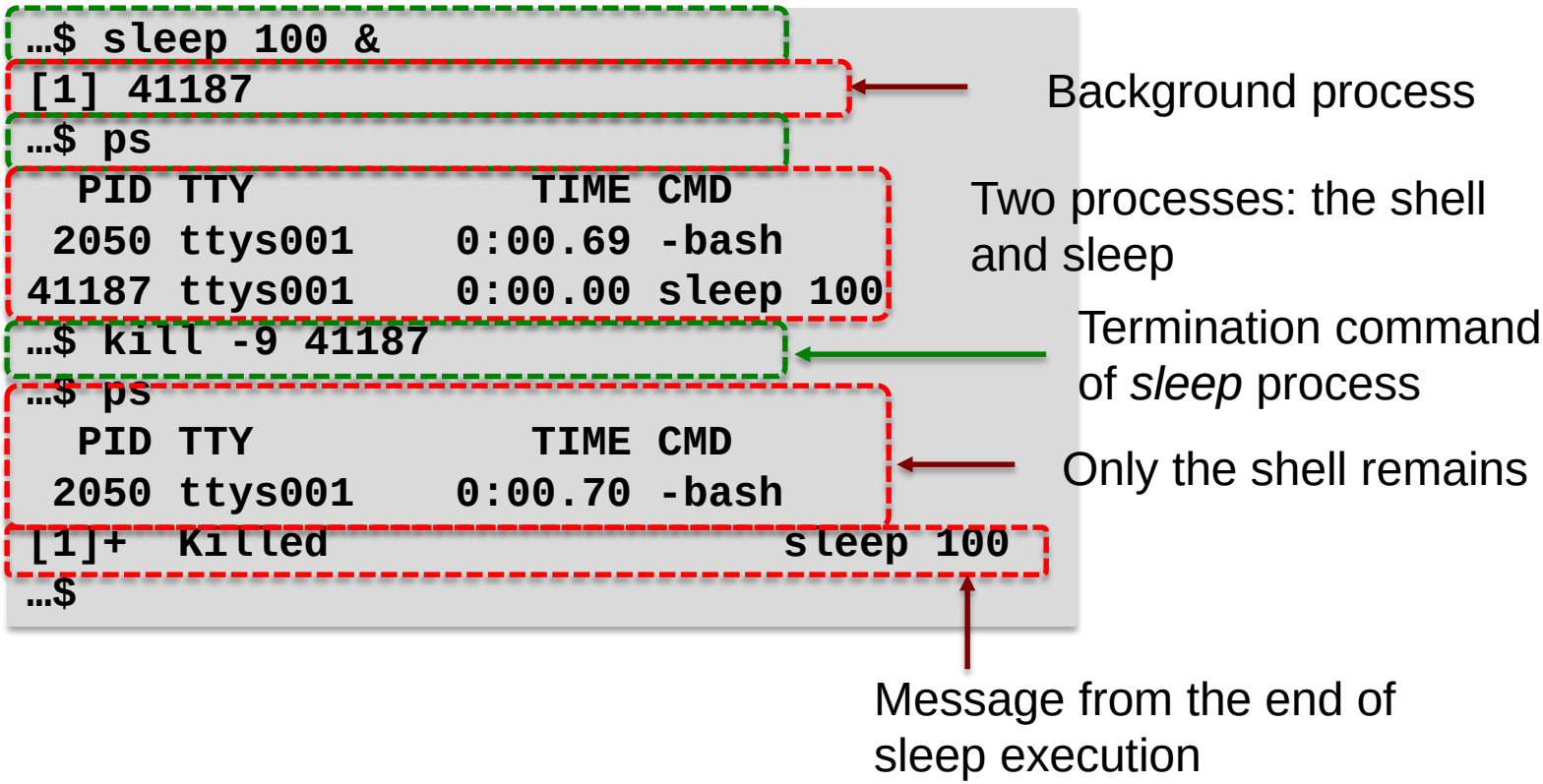
  - It shows parent & child relationship between processes
    - System start up (PID 0) ➔ init (PID 1) ➔ kdeinit (PID 19892) ➔ bash (PID 19894) ➔ ps -ef (PID 19914)

- Foreground and background execution modes
  - The SO creates a new process (shell child) executing an external command
  - Foreground (default): the shell waits the child process to end before continuing its running showing the prompt
    - The process can be stopped with ctrl-C (^ C)
  - Background: the shell and the child process execute concurrently
    - Sintax: ***command &***
    - The shell can stop the child process with `kill -9 PID`





- Examples


…$ sleep 100 &
[1] 41187
…$ ps
PID TTY TIME CMD
2050 ttys001 0:00.69 -bash
41187 ttys001 0:00.00 sleep 100
…$ kill -9 41187
…$ ps
PID TTY TIME CMD
2050 ttys001 0:00.70 -bash
[1]+ Killed sleep 100
…$
Background process
Two processes: the shell and sleep
Termination command of *sleep* process
Only the shell remains
Message from the end of sleep execution

Content

- Introduction
- Users and groups
- Shell variables
- Browsing the file system
- File system management
- Input / output management
- Process management
- **Shell programming**

Shell programming

- *Shell scripts*
  - They are text files containing shell commands
  - They accept arguments
  - They are executed
    - Using command **sh**:

      `sh script_directory_path/script_name` *`argument(s)`*
    - Directly by their name
      - Execution permission (x) have to be set
      - The script path has to be indicated if it is not on the PATH variable, in case of the working directory the prefix ./ have to be used:

      `$ ./script_name`
  - They inherit the environment of the shell
  - The script arguments are available inside the script as variables:

Symbol	Value
$0	Script name
$1…$9	1st … 9th argument
$#	Arguments number

```
#!/bin/sh
if[$# -gt 0 ]
then
 echo “param1 is $1”
else
 echo “Use $0 param1”
fi
```

Shell programming

- Exit code
  - The exit code is a numeric value returned by every command
    - $? gives the exit code value
  - Exit code 0: execution without errors
    - 0 = true if interpreted as a condition, any other value is false
  - Exit code 1.. 255: execution error

```
…$ ls
lote        lote-llarg…
…$ echo $?
0
…$ ls altre-nom
ls: altre-nom: No such file…
…$ echo $?
1
…$
```

- Command `exit` ends the script execution
- Some commands related to exit code

Command	Operation
`true`	Does nothing and it returns 0 as termination code
`false`	Does nothing and it returns 1 as termination code
`test`	Evaluates a condition, it returns termination codes 0 if true and 1 otherwise

- Examples

```
## createdir: script with one argument
## creates an empty directory with name given by argument $1

# If no file or directory exists with the given name, it is created
if ! test -e $1; then mkdir $1; exit 0;
# If the directory exists, its content is deleted
elif test -d $1; then rm -r $1/*; exit 0;
# Otherwise, do nothing and indicate error
else echo $1 already exitst and it is not a directory;
exit 1;
fi
```

```
## allold: script without arguments
## Adds ".old" at the end of all file names
## on the current directory

# i is a script local variable that takes values from the
# file names on the current directory
for i in * ; do mv $i $i.old; done
exit 0
```

```
$ for((i=1000;i--; i>0)); do echo “$i beers. Only $i beers
remain”; done
```

Exercises

- **Exercise SUT1.1_** Execute one by one the following shell command lines in a UNIX system, then analize the execution result of every one and answer the questions that follow:

```
$ cat test
$ echo “Hello I am an FSO student” >&1
$ echo “Hello I am an FSO student” >test
$ cat test
$ echo “I am practicing with the shell” >> test
$ cat  test | wc -l
```

Answer for every command line:

a) What is the execution result?

b) It has been created a new file?

c) Has been the command execution correct?

d) What returning codes has given back the system?

e) How many commands are included in the command line?

f) How many files intervene in the command line?

WARNING the character $ at the beginning of every command is the UNIX prompt, SO YOU DON’T HAVE TO TYPE IT.