

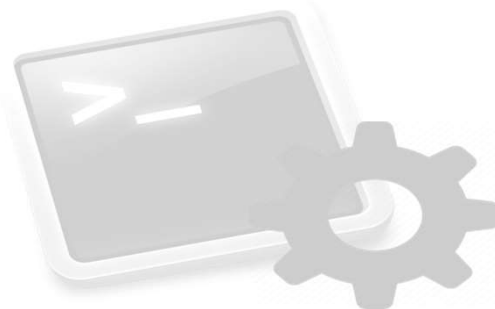
Command Line (shell)



FYI: Cartoons from the reference book, created by Lisa Haney (great artist).
portfolio: <http://lisahaney.com>.

Index

- **Shell Fundamentals**
- **String Manipulation**
 - Basic Commands
 - Regular expressions and grep
 - sed
 - awk
- **Shell Scripting**
 - Execution
 - Input/Output
 - Variables & Functions
 - Control Flow & Loops
 - Arithmetic
- **Shell Scripting (Python)**
- **Terminal Multiplexing**



- options> [arguments]

- **Options:** command pieces that modify the initial behavior
- **Arguments:** file name or any other kind of data needed by the command.

3



File Commands	System Info
ls - directory listing	date - show the current date and time
ls -al - formatted listing with hidden files	cal - show this month's calendar
cd dir - change directory to dir	uptime - show current uptime
cd - change to home	who - display who is online
pwd - show current directory	whoami - who you are logged in as
mkdir -dir - create a directory	finger - display information about user
rm file - delete file	uname - show system information
rm -r dir - delete directory	cat /proc/cpuinfo - cpu information
rm -f file - force remove file	cat /proc/meminfo - memory information
cp -r dir1 dir2 - copy dir1 to dir2	man command - show manual for command
cp file1 file2 - copy file1 to file2	df - show disk usage
cp -r dir1 dir2 - copy dir1 to dir2; create dir2 if it doesn't exist	du - show directory space usage
mv file1 file2 - rename or move file1 to file2	free - show system swap usage
if file1 is an existing directory, moves file1 into directory	whereis app - show possible locations of app
ln -s file link - create symbolic link link to file	which app - show which app will be run by default
ln -x file - create or update file	
cat > file - places standard input into file	Compression
more file - view file, one screen at a time	tar cf file.tar files - create a tar named file containing files
head file - output the first 10 lines of file	tar xf file.tar - extract the files from file.tar
tail file - output the last 10 lines of file	tar czf file.tar.gz files - create a tar with Gzip compression
zcat file - output the contents of file as it grows, starting with the last 10 lines	tar xzf file.tar.gz - extract a tar using Gzip
	tar cjf file.tar.bz2 - create a tar with Bzip2 compression
	tar xjf file.tar.bz2 - extract a tar using Bzip2
	gzip file - compresses file and renames it to file.gz
	gunzip file.gz - decompresses file.gz back to file
Process Management	Network
ps - display your currently active processes	ping host - ping host and output results
top - display all running processes	whois domain - get whois information for domain
kill pid - kill process with pid	nslookup domain - get DNS information for domain
killall proc - kill all processes named proc *	nslookup - get host - reverse lookup host
bg - lists stopped or background jobs; resume a stopped job in the background	wget file - download file
fg - brings the most recent job to foreground	wget -c file - continue a stopped download
fg - brings job n to the foreground	
	Installation
File Permissions	Install from source:
chmod octal file - change permissions of file to octal, which can be found separately for user, group, and world by adding:	./configure
<ul style="list-style-type: none"> • 4 - read (r) • 2 - write (w) • 1 - execute (x) 	make
Examples:	make install
chmod 777 - read, write, execute for all	dpkg -i pkg.deb - install a package (Debian)
chmod 755 - rwx for owner, rx for group and world	rpm -Uvh pkg.rpm - install a package (RPM)
For more options, see man chmod .	
	Shortcuts
SSH	Ctrl+C - halts the current command
ssh user@host - connect to host as user	Ctrl+Z - stops the current command, resume with fg to return to foreground
ssh -p port user@host - connect to host on port port as user	Ctrl+D - log out of current session, similar to exit
ssh -p port id@host - add your key to host port to enable a key or passwordless login	Ctrl+W - erases one word in the current line
	Ctrl+U - erase the current line
Searching	Ctrl+R - type to bring up a recent command
grep pattern files - search for pattern in files	!! - repeats the last command
grep -r pattern dir - search recursively for pattern in dir	exit - log out of current session
command grep pattern - search for pattern in the output of command	
cat file grep pattern - find all instances of file	
	* use with extreme caution

- An example Cheatsheet, look for the one that fits best for your needs (google).

The shell Fundamentals (Configuration)

- **Environment Variables:** shell session variables with a pre-defined value.
 - Their value is obtained this way: `$ echo $VARIABLE`
 - Two kinds:
 - **User variables:** internal to our shell session, can be listed with command `env`.
 - **System variables:** common to every shell, programs and users.
 - Environment variables can be modified: `setenv (csh, tcsh)`, `export (sh, bash)`.
 - Example: `$ export PATH=/usr/local/bin:/bin:/usr/bin`
 - After leaving a session, all modifications are lost.
 - Some important internal variables:
 - **\$PATH:** indicates which are the directories where binaries can be found. Before executing a command, the shell searches in those directories.
 - **\$HOME:** root directory of current user.
 - **\$TERM:** kind of terminal we are employing to connect to the system.
 - **\$SHELL:** User shell. Ex. `/bin/bash`
 - **\$****:** In the man page of each shell we have the complete repertory of its environment variables.

The shell Fundamentals (Configuration)

- **Shell configuration files:** permanent value for some environment variables
 - Bash loading sequence depends on the kind of shell (last file overwrites the rest)
 - **Interactive + login:** `/etc/profile` → `$HOME/.bash_profile` (or `.bash_login` or `.profile`)
 - **Interactive + non-login:** `/etc/bash.bashrc` → `$HOME/.bashrc`
 - File example(bash):

The alias command allows command re-definition (more friendly shell)

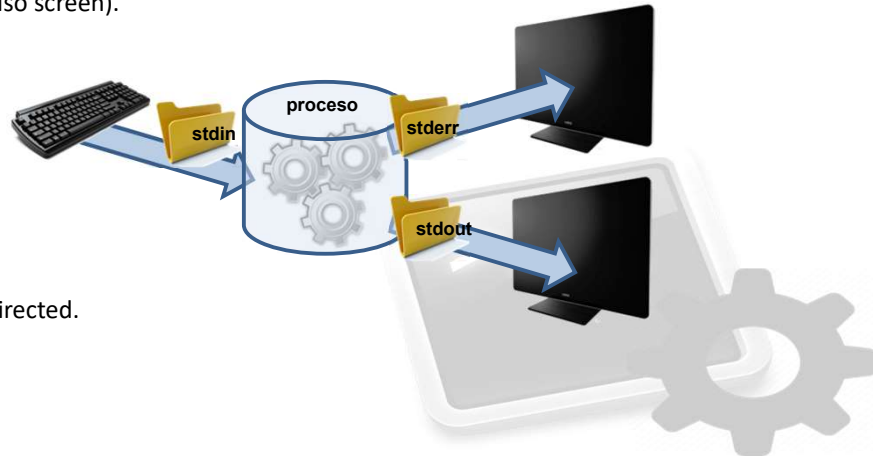
```
# .bashrc
# User specific aliases and functions
alias rm='rm -i'
alias cls="clear"
alias cd.='cd ..'

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

alias ls="ls --color -shaF"
```

The Shell: Pipes and Redirection

- In linux, always three files (devices treated as files) opened by default: **stdin** (keyboard), **stdout** (screen) and **stderr** (also screen).
- By default:

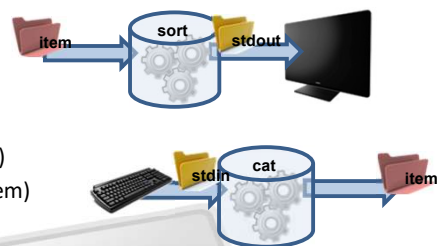


- These files can be redirected.

The Shell: Pipes and Redirection

- **Redirection**: modify standard input/output for a command.

- Standard **input** redirection: do not use keyboard as input.
- Syntax: `$ sort < item`
- Standard **output** redirection: output to a file (instead of the screen)
- Syntax: `$ ls -l > item` (without overwriting item content: `$ ls -l >> item`)
 - `1>` (redirect stdout, same as default) `2>` (redirect stderr)
 - `2>&1` (redirect both to same location)



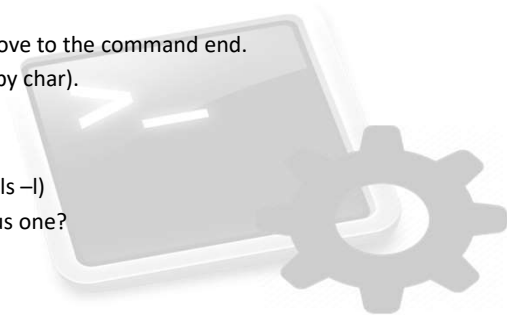
- **Pipes**: Allows two or more command linking, where the output of a command is redirected to be the input of the following one.

- Example: `cat /etc/passwd | grep root | cut -d : -f 7 > root_shells`



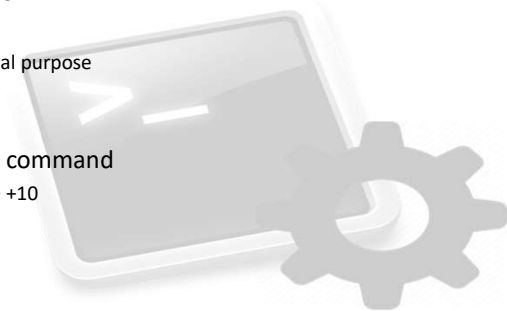
The Shell: Shortcuts (To make your life easier)

1. commands/filenames/paths can be **autocompleted** (Tab in bash)
 - If it cannot be completely resolved, a list with all the alternatives is displayed.
2. Special movements using the **cd command**:
 - **[cd -]** will put you back in the last working directory.
 - **[cd or cd ~]** will put you back in your \$HOME directory.
3. Moving the **cursor** through the command line (prompt)
 - **[Ctrl+a]**: go to the beginning of the command. **[Ctrl+e]**: move to the command end.
 - Cursor Left/Right: move through the command line (char by char).
 - **[Ctrl left/right]**: move word by word.
4. **Concatenate** command execution in the same line.
 - Example: `ls -l; cd ..; ls -l` (also this way: `ls -l && cd .. && ls -l`)
 - **nested** execution: `(ls -l; cd ..); ls -l` Difference with previous one?
5. Use **alias** for frequent commands and to fix typos
 - `alias gerp=grep`, `alias ll=ls -la`



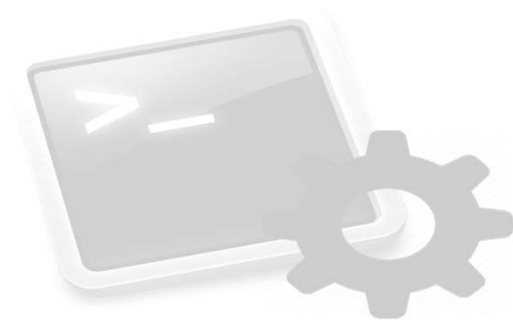
The Shell: Shortcuts (To make your life easier)

6. **Search and reuse** previous commands
 - The commands employed in a shell session are stored. With the command history we can review commands executed, repeat or edit previous commands.
 - **!!**: execute again the last command of the list (previous command)
 - **!letters**: execute again the last command executed starting with the indicated letters
 - **!number**: execute the command in the list with that number.
7. Employ **wildcards**.
 - Some characters cannot be employed in filenames, having a special purpose
 - **"*"**: replace all characters: `$ ls -l pa* // $ rm -fr /*` (oops!!)
 - **"?"**: replace a single
8. Run the same command over a list of arguments: **xargs** command
 - Example: `$ ps -ef | grep "pepito" | awk '{print $2}' | xargs renice +10`
9. **Scripting**, scripting and more scripting...



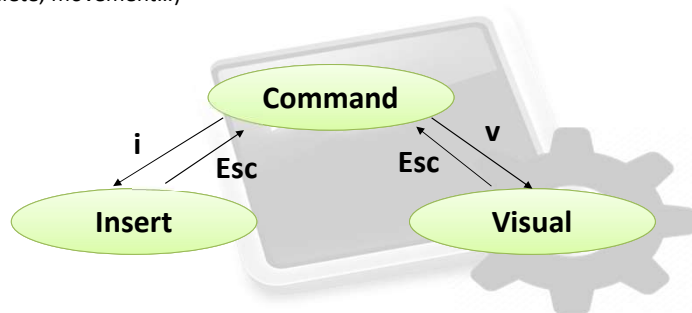
Index

- Shell Fundamentals
- String Manipulation
 - Basic Commands
 - Regular expressions and grep
 - sed
 - awk
- Shell Scripting
 - Execution
 - Input/Output
 - Variables & Functions
 - Control Flow & Loops
 - Arithmetic
- Shell Scripting (Python)
- Terminal Multiplexing

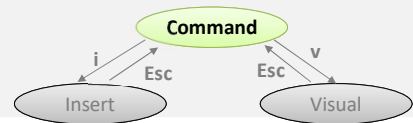


String Manipulation

- File edition with **vi/vim**: terminal text editor included in every UNIX system.
 - A bit awkward for begginers. With time and practice, much faster than any graph editor.
- 3 Operation modes:
 - Command: define operations (copy, delete, movement...)
 - Insert: write file content (text)
 - Visual: text selection.



String Manipulation



- File edition with **vi/vim**: a simple cheatsheet (**command operation mode**).

Open, Save+Quit	
:q	quit
:q!	Quit, throw away changes
:w name	Save as name
:x	Quit + save
:e name	Open file <name>

Movement (Horizontal)	
→, l	Right one character
←, h	Left one character
0	Beginning of line
\$	End of line
w/W	Beginning of next word/bigword
e/E	End of next word/bigword
b/B	Beginning of previous word/bigword

Movement (Vertical)	
↑, k	Up one line
↓, j	Down one line
PgUp, ^B	Up one page
PgDn, ^F	Down one page
(num)G	Go to line (num)
/string	Find string
n/N	Repeat search, fward/backward.

Basic edits
these commands modify text but keep you in command mode

x	Delete char under cursor
J	Join next line to end of current line
r(char)	Replace char under cursor with (char)
dd	Delete current line
dw	Delete current word
d(move)	Delete from cursor to (move)
yy	Copy current line
yw	Copy current word
u	Undo!
.	Repeat last edit command

Compound commands (the power of vi)

3→	3 chars right
4)	4 sentences right
2b	2 words left
d3w	Delete 3 next words
d)	delete reminder of paragraph
d)	delete reminder of sentence

Insert Mode

i/a	Insert before/after cursor
I/A	Insert at beginning/end of line
o/O	New line below/above, then insert
cc	Replace current line
c(move)	Replace to (move)

Search & Replace

:s/reg/rep/	1 st match, current line
:s/reg/rep/g	All matches, current line
:%s/reg/rep/g	Global replace

Windowing

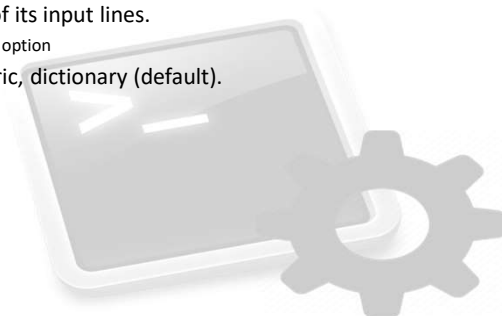
:sp file	new split-frame window
^w^w	go to next window

Cut + Paste

- 1.- Press v to enter visual mode
- 2.- Move cursor to highlight text
- 3.- Press d to cut, y to copy
- 4.- Move to target location
- 5.- Hit P/p to paste after/before cursor.

String Manipulation

- One of the most common tasks in system administration.**
 - Almost everything, from report writing, to printing mailing labels, to editing documents, to create administration scripts, to configure the environment, relies on string manipulation.
- Common filter Commands for data processing(extended in appendix):**
 - cut:** separate lines into fields and print selected portions of its input lines.
 - The default delimiter is <tab>, but you can change delimiters with -d option
 - sort:** sort input lines. Multiple sorting mechanisms: numeric, dictionary (default).
 - Example: sort -t: -k3 -n /etc/group
 - uniq:** print unique lines.
 - Example: cut -d: -f7 /etc/passwd | sort | uniq -c
 - wc:** count lines, words and characters.
 - head** and **tail:** read the beginning and end of a file.



String Manipulation

- Command **sed**: perform line by line text modifications in an input file.
 - **Syntax**: `sed -<OPTS> '[COMMAND]' [INPUTFILE]`
 - **OPTS**: modify default behavior
 - Option `-i`: in place, the file passed as argument is overwritten.
 - **COMMAND**: each one with the same structure: `'[FILTER]X[OPTS]'`
 - **FILTER**: Execute the command only in matching lines. Can be a line number, a range of lines or a matching regular expression.
 - **X[OPTS]**: defines the action to perform on inputfile.
 - `a/`: append after / insert before a line.
 - `p`: print current line in stdout.
 - `c`: replace lines with text.
 - ... (man sed for the rest of actions)
 - **Some Examples**:
 - `sed 'a\mytext' infile` ; `sed 'a mytext' infile` ; `sed '1,3a\mytext' infile` ; `sed '2,3 p' infile` ; `sed '3 q' infile`
 - **The s Command (substitute): replace text, syntax: '[FILTER]s/regexp/replacement/flags'**
 - Match the pattern of the regular expression, if successful replace with replacement
 - `sed -i 's/Pepe/Manolo/g' *.txt` replace pepe by manolo in every .txt file
 - `sed '/cadena/ s/vieja/nueva/g' file > salida` only replace in lines containing the string (flag `g`: perform the change in every matching).

Want more examples?

<https://www.thegeekstuff.com/tag/sed-tips-and-tricks/>
And infinite more (google: sed examples administration)

String Manipulation

- The command **grep**: searches its input text and prints the lines matching a given pattern. Many options available (man grep)
- **Regular Expressions**:
 - Employed to match a text string to a pattern (semi-generic pattern)
 - Simplest regexp: literal characters: `a` matches `a`, `AR` matches `AR`.
 - Literal combination: `PA | MA` matches `PA` & `MA`.
 - basic (default) vs. extended (`-E`)
 - Pattern: built through a mix of literal and special characters.
- **Lists of characters**:
 - Syntax: `[list]`, matches any single character in the list
 - Example: `[12345689]` matches any single digit except 0 or 7
 - `[^]`: non-matching list
 - `[3-7]`: ranged expressions (from 3 to 7)
 - Named classes available: `[[:alnum:]]`, `[[:alpha:]]`, `[[:digit:]]`, etc.

String Manipulation

- The command **grep**: searches its input text and prints the lines matching a given pattern. Many options available (man grep)

- Metacharacters:**

in basic syntax, some metacharacters must be preceded by "\ " (man grep)

Sym	What it matches or does
.	Matches any character
^	matches the beginning of a line
\$	matches the end of a line
\w	matches any alphanumeric character ([A-Za-z0-9_])
?	Allows zero or one match of the preceding element
*	allows zero, one or many matches of preceding element
+	allows one or more matches of preceding element
{n}	Matches n instances of preceding element
{n,}	Matches n or more instances of preceding element
{n,m}	Matches n to m instances of preceding element

- Subexpression:**

- The back-reference '\n', where n is a single digit, matches the substring previously matched by the nth parenthesized subexpression of the regular expression.
- Example: '(a)\1' matches 'aa'
- \$ grep "\([aeiou]\)\.1" input

String Manipulation

- awk** programming language: oriented to file processing.
 - Line-oriented (file is analyzed line-by-line).
 - Not oriented to text modifications (like sed), no overwrite available.
 - Basic format of an awk program is: **awk [options] '/pattern/ { action }' file**

- pattern determines when to perform action.
- If pattern condition returns true, in that line action is performed.
- If pattern is left empty, action is performed in every line.
- awk variables**
 - \$N: this var contains the N field of the line (default field separator: space)
 - \$0: variable containing the whole line
 - FS: determines a different field separator (option -F)
 - NF: Contains the number of fields in a line
 - NR: Contains the line number.

- Examples:**

- awk -F: '{if(\$2=="") print \$1": no password!"}' /etc/passwd
- awk '{ if(NR>100) print NR, \$0}' fichero

Want to learn all awk possibilities?
Take a look at the manual (a whole book)
<https://www.gnu.org/software/gawk/manual/gawk.html>

Index

- **Shell Fundamentals**
- **String Manipulation**
 - Basic Commands
 - Regular expressions and grep
 - sed
 - awk
- **Shell Scripting**
 - Execution
 - Input/Output
 - Variables & Functions
 - Control Flow & Loops
 - Arithmetic
- **Shell Scripting (Python)**
- **Terminal Multiplexing**

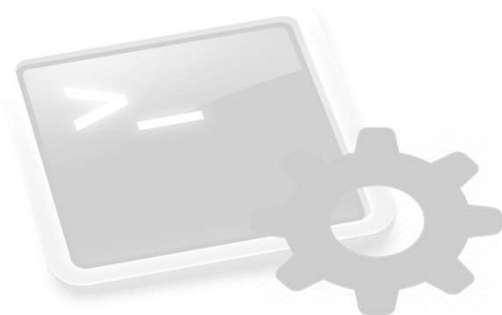
www.ce.unican.es/SI/LabFiles/scripts-ejemplo.tar.gz



Shell Scripting

- Group shell commands to perform complex tasks in a single step.
- The simplest structure: text file with a command per line, but usually much more complex
 - Conditional sentences, loops, functions,...
- A script creates a sub-shell independent to the one that executed it.
 - \$ bash macro
- Its structure depends on the shell we are employing
- Example:

```
#!/bin/sh
echo "Today is:"
date
echo "have a nice day"
```
- First line: shebang interpreter directive
 - indicates which kind of shell executes the rest of the script.
 - by default (when left empty) /bin/sh.



Shell Scripting

```
# run ./script [name]
#!/bin/sh
echo -n "Your surname? "
read surname
echo "Hola, $1 $surname"
```

- **Execution:**
 - Launch the script as a program : `$ /bin/bash script` (Or modify permissions: `chmod +x script`).
 - Source the script as a “bunch of text”, processed by current shell (`source script.sh`)
- **Variables and quoting:**
 - unmarked in assignment but prefixed with “\$” when referenced:
 - `etcdir=/etc`; `echo $etcdir` (**Omit spaces around =**, otherwise, the variable name is taken as a command).
 - Quotation marks: Command substitution (output of a command stored in a variable)
 - Single quotes ('): string definition, holds literal value of each character within the quotes
 - Double quotes ("): string definition, allows to interpret dollar (\$), backtick (`), backslash (\) and exclamation (!).
 - Backtick (`): interpret the command inside `var1=`command`` (watch out the kind of angle bracket, ` works, " not)
- **Input/Output:**
 - Read from keyboard with command **read**.
 - Write in screen with **echo/printf**.
 - Redirect stdout and stderr as seen previously.

comillas.sh

Shell Scripting

- **Command line arguments**
 - Positional Parameters: become variables whose names are numbers
 - `$1 - 9`: Command line parameters, number indicates its position
 - `$0`: macro name(script name)
 - `$#`: number of command line argument.
 - `$?`: `$$`: PID associated to the macro.
 - `$*`: string containing all the arguments passed (beginning with `$1`)
 - Flags: single letters starting with a hyphen (-) before each argument (like command options)

flags.sh

```
while getopts u:a: flag
do
    case "${flag}" in
        u) username=${OPTARG};;
        a) age=${OPTARG};;
        esac
    done
```

Shell Scripting

```
if [ $foo -ge 3 ]; then
    echo "Variable larger than 3"
else
    echo "Var below 3"
fi
```

- Control Flow with if-then-(elif)-else:

- Three types of conditions: file-based, string-based and arithmetic.
 - The condition is essentially a command. `if [$foo -ge 3]; then` is equivalent to `if test $foo -ge 3; then`
- Basic rules of conditions:
 - Always keep spaces between the brackets and the actual check/comparison
 - Always terminate the line before putting a new keyword like "then"
 - It is a good habit to quote string variables if you use them in conditions

- comparison operators

String	Numeric	True if
<code>x = y</code>	<code>x -eq y</code>	x is equal to y
<code>x != y</code>	<code>x -ne y</code>	x not equal to y
	<code>x -lt y</code>	x is less than y
...	le, gt, ge	...

- ... and Bash file evaluation operators

Operator	True if
<code>-d file</code>	file exists and is a directory
<code>-e file</code>	file exists
<code>-f file</code>	file exists and is a regular file
<code>-r file</code>	you have read permission on file
<code>-s file</code>	file exists and is not empty
<code>-w file</code>	you have write permission on file

compare.sh

- Double bracket syntax: enhanced version

- <https://linuxacademy.com/blog/linux/conditions-in-bash-scripting-if-statements/>

```
if [[ $(who | grep -s pepe) > /dev/null ]]; then
    echo "pepe is in the system"
fi
```

Shell Scripting

- Using Case statements:

- `case` EXPRESSION in CASE1) COMMAND-LIST;; CASE2) COMMAND-LIST;; ... CASEN) COMMAND-LIST;; esac

```
#!/bin/bash
# This script does a very simple test for checking disk space.

space=`df -h | awk '{print $5}' | grep -v Use | sort -n | tail -1 | cut -d "%" -f1`

case $space in
[1-6]*)
    echo "All is quiet."
    ;;
[7-8]*)
    echo "Start thinking about cleaning out some stuff. A partition is $space % full."
    ;;
9[1-8])
    echo "Better hurry with that new disk... One partition is $space % full."
    ;;
99)
    echo "I'm drowning here! There's a partition at $space %!"
    ;;
*)
    echo "I seem to be running with an nonexistent amount of disk space..."
    ;;
esac
```

Shell Scripting

```
for archivo in $(ls); do
    touch $archivo
    echo "archivo $archivo update"
done
```

loops.sh

- The **for** Loop:
 - List of arguments: for files in fich1.sh fich2.sh fich3.sh; do
 - Pattern matching expansion: for files in *.sh; do
 - Command outputs the list: for files in \$(ls); do
- The **while** loop:
 - Useful for processing command-line arguments and reading the lines of a file

```
while IFS= read -r line do
    echo "$counter: $line"
    counter=$((counter + 1))
done < /path/to/file
```

Arithmetic:

- All bash variables are string valued
 - Declaration: a=pepe, Utilization: echo "\$a"
- Arithmetic with variables?: (())
 - operation in (()) is arithmetic, otherwise only concatenated strings
- Arrays:
 - Declaration: list=(aa bb cc dd), Utilization: echo "\${list[num]}" (begins in zero)

numeric.sh

```
a=1
b=$((2+2))
c=2+2

n1=$a+$b+$c
n2=$a$b$c
n3=$(( $a+$b ))
echo "$n1 $n2 $n3"
```

Shell Scripting

- The **for** Loop: alternative syntax and compatibility

Type	syntax	Bash-compatible	Sh-compatible
Numeric range	for i in 1 2 3 4 5	Yes	Yes
Numeric range	for i in {1..5}	Yes	No
Numeric range	for i in {0..10..2}	Yes	No
String range	for i in a b c d e	Yes	Yes
Command output	for i in \$(ls *.txt)	Yes	Yes
C-like	for ((i=1; i<=5; i++))	Yes	No

Shell Scripting: my advice for exams

- Take an **incremental approach**, never look for the whole solution from scratch.
- Example (Lab exercise): "Write a script that asks the user to type a word and checks if that is an available user command or not".

```
# Step 1: prepare input/output
#!/bin/bash
```

```
echo -n "Please
read candidate
echo ""
echo "your string"
```

```
# Step 2: sketch your control flow
```

```
#!/bin/bash
candidate="ls"
while [ $candidate != "quit" ]; do
    echo -n "Please type your command: "
    read candidate
    if [[ $candidate = "ls" ]]; then
        echo "your string is $candidate"
    else
        echo "not ls"
    fi
done
echo "Leaving"
```

```
# Step 3: google "check if shell command exists"
# type / command / which
```

```
#!/bin/bash
candidate="ls"
while [ $candidate != "quit" ]; do
    echo -n "Please type your command: "
    read candidate
    if [[ $(command -v $candidate) ]]; then
        echo "el comando SI existe"
    else
        echo "el comando NO existe"
    fi
done
echo "Leaving"
```

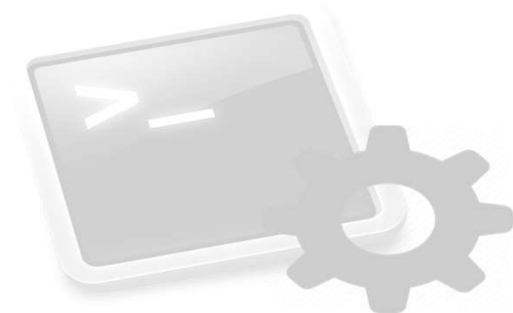
Sistemas Informáticos

27



Index

- Shell Fundamentals
- String Manipulation
 - Basic Commands
 - Regular expressions and grep
 - sed
 - awk
- Shell Scripting
 - Execution
 - Input/Output
 - Variables & Functions
 - Control Flow & Loops
 - Arithmetic
- Shell Scripting (Python)
- Terminal Multiplexing



Sistemas Informáticos

28



Shell Scripting: Python



- Python is a full-fledged programming language that can be easily employed as a scripting language
 - In fact, it is possible to implement platform independent scripts, even if they access OS specific functionalities.
- Not sure about your skills with Python, so this part will not be included in evaluation.
 - Use it if you feel confident about it. It won't be mandatory to use python during an exam.
- Python provides various modules to interact with the Operating System:
 - **os**
 - **shutils**
 - **subprocess**
 - **sys, glob...**



Shell Scripting: Python



- **os module:** functions for interacting with the operating system.
 - comes under Python's standard utility modules (no "pip install" required).
- Many functions to interact with the **file system**:
 - **os.getcwd()**: Return a string representing the current working directory.
 - **os.chdir(path)**: Change the current working directory to *path*.
 - **os.mkdir(path, mode=0o777)**: Create a directory named *path* with numeric mode *mode*.
 - **os.listdir(path)**: returns a list containing the names of the entries in *path*
 - ...
- **Environment management:**
 - **os.getenv(key, default=None)**: return the value of env variable key (default if it does not exist)
 - **os.putenv(key, value)**: set an environment variable
- **Process management:**
 - **os.getpid()**: return current process id
 - **os.nice(increment)**: add increment to process niceness
 - **os.kill(pid, sig)**: send signal sig to process pid
- ...

```
#!/usr/bin/env python3
# program to change the working directory

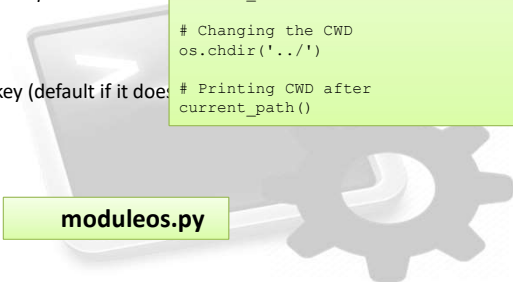
import os
def current_path():
    print("Current working dir")
    print(os.getcwd())

# Printing CWD before
current_path()

# Changing the CWD
os.chdir('../')

# Printing CWD after
current_path()
```

moduleos.py



Shell Scripting: Python



- **shutil module**: high level operations on a file (create, copy, etc.)
 - comes under Python's standard utility modules (no "pip install" required).
- **shutil.copy(src, dst, *, follow_symlinks=True)**:
 - copy src file to dst. if dst is a directory, src name is used for destination file.
 - if "follow_symlinks" is false and src is a symlink, dst will also be a symlink.
 - copies data and permissions, but not metadata (use copy2 instead).
- **shutil.move(src, dst, copy_function=copy2)**:
 - move recursively a file/directory from src to dst.
- **shutil.copytree/shutil.rmtree(opts)** :
 - copy (remove) recursively a directory indicated by src to dst.
- **shutil.make_archive(name, format, root_dir, ...)**:
 - create a container file (tar, zip, gz) and return its name.
 - Available formats: zip, tar, gz-tar, bztar, xztar.
- **shutil.unpack_archive(filename, extract_dir, format)**
 - unpack a container file.

moduleshutil.py

```
#!/usr/bin/env python3
import os, shutil

archive_name = os.path.join('/root', 'myarchive')
root_dir = os.path.join('/root', '.ssh')
shutil.make_archive(archive_name, 'gztar',
root_dir)
```

Shell Scripting: Python



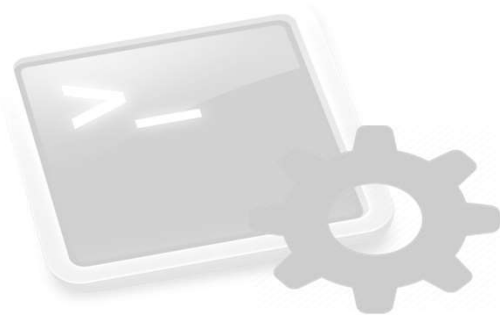
- **subprocess module**: create spawn processes with the module subprocess, connect to their input, output, and error pipes, and obtain their return codes.
 - **subprocess.run(args, *, stdin=None, input=None, stdout=None, stderr=None, capture_output=False, shell=False, ...)**
 - returns a "CompletedProcess" object.
- Use always shell=False
- Command output can be redirected.
 - To the CompletedProcess object: stdout=subprocess.PIPE stderr=subprocess.PIPE
 - Ignored: stdout=subprocess.DEVNULL if capture_output is true, stdout and stderr of the command will be captured
- If check is true, when the process exits with a non-zero code (fails), an exception is raised.

modulesubprocess.py

```
#!/usr/bin/env python3
import subprocess
out = subprocess.run(["ls", "-l"], shell=False)
print(out)
```


Index

- Shell Fundamentals
- String Manipulation
 - Basic Commands
 - Regular expressions and grep
 - sed
 - awk
- Shell Scripting
 - Execution
 - Input/Output
 - Variables & Functions
 - Control Flow & Loops
 - Arithmetic
- Shell Scripting (Python)
- Terminal Multiplexing



Terminal Multiplexing

- **Terminal Multiplexing:** open multiple windows and split-views within one terminal.

```
1 [[[[[ 0.3%]]] 5 [[[[[ 21.0%]]]
2 [[[[[ 65.2%]]] 6 [[[[[ 44.8%]]]
3 [[[[[ 14.0%]]] 7 [[[[[ 44.6%]]]
4 [[[[[ 15.9%]]] 8 [[[[[ 14.2%]]]
Mem[[[ 1641/4013MB]]] Tasks: 260; 5 running
Swap[[[ 2/8181MB]]] Load average: 0.79 0.28
Uptime: 6 days, 11:15:24

PID USER      CPU% MEM% Command
14574 root      0.0 12.7 /usr/lib/jvm/java-7-openj
14521 root      0.0 12.4 /usr/lib/jvm/default-java
907 root      21.3 3.5 /usr/lib/jvm/default-java
12495 root      0.0 1.0 /usr/sbin/mysqld --based
11911 root      0.0 0.7 /usr/sbin/named -u bind
17123 root      0.0 0.1 vim
15381 root      0.0 0.1 vim /etc/varnish/default
F1Help F2Setup F3Search F4Filter F5View F6Build F7

nineteen@manager:~/dev/nineteen/src/server$ grails
Configuring classpath

35 repositories {
36   inherits true // Whether to inherit repository definition
   s from plugins
37
38   grailsPlugins()
39   grailsHome()
40   mavenLocal()
41   grailsCentral()
42   mavenCentral()
43
44   // uncomment these (or add new ones) to enable remote dep
   endency resolution from public Maven repositories
45   //mavenRepo "http://repository.codehaus.org"
46   //mavenRepo "http://download.java.net/maven/2/"
47   //mavenRepo "http://repository.jboss.com/maven2/"
48
49   dependencies {
50     // specify dependencies here under either 'build', 'compi
   le', 'runtime', 'test' or 'provided' scopes e.g.
51   }
52
53   plugins {
54     // plugins for the build system only
55     build "com.ea:ant-tasks:1.5.1"
56
57     // plugins for the compile step
58     compile "org.hibernate:hibernate-core:3.6.10.2"
59     compile "org.quartz-scheduler:quartz:2.2.1"
60
61     // plugins needed at runtime but not for compilation
62     runtime "org.springframework:spring-core:3.1.1"
63   }
64 }
```

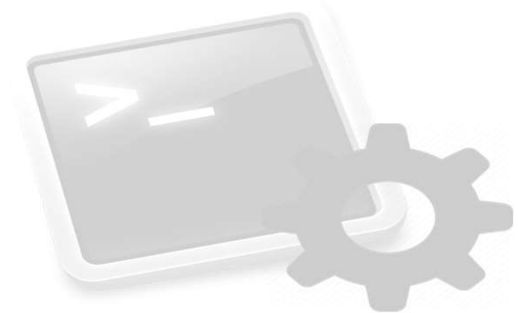
Terminal Multiplexing

- **TMUX:** (<https://github.com/tmux/tmux>)
 - Starting a session: command `tmux`.
 - All-green status bar at the bottom holds session information: opened windows (left), date & time (right)
 - `tmux` commands: prefix key [`Ctrl+b`] followed by command key
 - `Ctrl+b` pressed at the same time, then released, then the command key is pressed.
 - Creating Windows:
 - `Ctrl-b c`: create a new window (check the status bar)
 - `Ctrl-b p/n`: move to the previous/next window
 - Managing panes:
 - `Ctrl-b %`: horizontal split. `Ctrl-b “`: vertical split
 - `Ctrl-b <arrow key>`: switch to a different pane.
 - `Ctrl-d` (or typing `exit`): Close a pane.
 - `Ctrl-b z`: make a pane go full screen. Type it again to shrink it back.
 - You can exit a session at any point (detach: `Ctrl-b d`) and `tmux` will keep this session alive (until you kill the `tmux` server). Useful to re-attach to exited sessions (`tmux ls`, `tmux attach -t [num]`).



Terminal Multiplexing

- **TMUX: Copy text between panes**
 1. `Ctrl-b [` : Enter copy mode
 2. Move to the beginning of the text to copy
 3. `Ctrl-SPACE`
 4. Move with the cursor until the end of the text to copy
 5. `Alt+w` : Copy the text into `tmux` clipboard
 6. Move to other pane
 7. `Ctrl-b]` : paste copied text from `tmux` clipboard



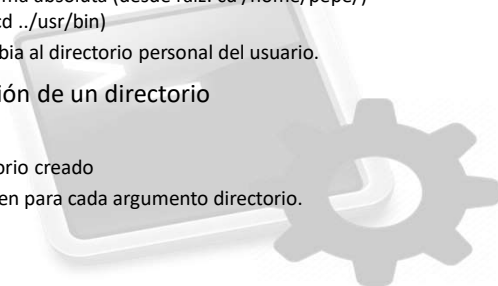
APPENDIX



Sistema de ficheros (Comandos)

Navegación por el sistema de ficheros

- Comando **pwd**: Imprime por pantalla el path del directorio donde nos encontramos.
- Comando **cd**: Comando utilizado para cambiar de directorio.
 - Generalmente, el inicio de sesión en el directorio personal del usuario.
 - Sintaxis: `$ cd [directorio]`
 - El directorio destino se puede expresar de forma absoluta (desde raíz: `cd /home/pepe/`) o de forma relativa (desde directorio actual: `cd ../usr/bin`)
 - Si no especificamos destino, el comando cambia al directorio personal del usuario.
- Comando **mkdir**: Comando para la creación de un directorio
 - Sintaxis: `$ mkdir -<opciones> directorio`
 - Opción **-m**: Establece los permisos del directorio creado
 - Opción **-p**: crea los directorios padre que falten para cada argumento directorio.



Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **ls**: Uno de los más utilizados. Lista el contenido de un directorio alfabéticamente.
 - Sintaxis: `$ ls -<opciones> [archivo...]`
 - Si se ejecuta sin argumentos, lista los archivos (y directorios) del directorio actual.
 - Opción **-a**: incluye los archivos ocultos (empiezan por `.`) al listar
 - Opción **-l**: lista archivos con detalle (permisos, enlaces, propietario, grupo, tamaño, fecha de modificación)
 - Opción **-r**: invierte el orden de listado de los archivos.
 - Opción **-t**: ordena la lista por tiempo de modificación.
 - Opción **-S**: ordena la lista por tamaño de archivo.
 - Opción **-s**: muestra el tamaño de cada archivo (en Kbytes) a la izquierda del nombre.
 - Opción **-A**: lista todos los archivos excepto `"."` y `".."`
 - Opción **-R**: lista los contenidos de todos los directorios recursivamente.
 - Opción **-color**=[none/auto/always]: emplear color para distinguir tipos de archivos.
 - Ejemplo combinado: `$ ls -lart` ¿Qué hace este comando?

Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **cp**: Utilizado para copiar archivos.
 - Sintaxis: `$ cp -<opciones> [arch_1]...[arch_n] [dir-destino]`
 - Opción **-f** (forced): Sobre escribe los archivos de destino con el mismo nombre.
 - Opción **-i** (interactive): lo contrario de **-f**, preguntar antes de sobrescribir.
 - Opción **-p**: Conserva los permisos, usuario y grupo del archivo a copiar.
 - Opción **-R**: Copia directorios recursivamente.
 - Opción **-a**: equivalente a **-pR**.
 - Opción **-u**: si en destino existe un archivo de mismo nombre y con igual o más reciente fecha de modificación, no se realiza la copia.
 - Opción **-v** (verbose): Imprime por pantalla información sobre lo que se copia.
- Comando **mv**: Comando utilizado para mover archivos (no copiar) y/o renombrarlos.
 - Sintaxis: `$ mv -<opciones> [origen_1]...[origen_n] [destino]`
 - Si el último argumento es un directorio, se mueven cada uno de los ficheros origen a ese dir.
 - Si origen y destino son ficheros, se renombra dicho fichero.

Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **rm**: Borrado de archivos o directorios (Peligro!!).
 - Sintaxis: `$ rm -<ops> [archivo]...`
 - Advertencia: Utilizar con cuidado. NO ES UN ENVÍO A LA "PAPELERA DE RECICLAJE".
 - El argumento [archivo] puede ser un archivo, un directorio o una expresión regular.
 - Opción **-f** (forced): Sin mensajes de error, sin solicitar confirmaciones (mucho ojo).
 - Opción **-r** (recursive): Borrar los contenidos de directorios recursivamente.
- Comando **ln**: Establece enlaces entre archivos
 - Se pueden crear tanto enlaces rígidos como simbólicos.
 - Sintaxis: `$ ln -<ops> [origen] [destino] $ln -<ops> [origen]... [directorio]`
 - Opción **-d**: permite al superusuario hacer enlaces rígidos a directorios.
 - Opción **-s**: crear enlace simbólico.
 - Ejemplo: `$ ln -s /etc/passwd /home/usuario/claves`
 - Cuando ejecuto `ls -l` en un directorio con enlaces simbólicos:
 - `lrwxrwxrwx 1 usuario usuario 11 Apr 8 13:33 claves -> /etc/passwd`

Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **whereis**: localizar path de binario/código fuente/manual de un comando.
 - Sintaxis: `$ whereis -<opciones> [archivo]...`
 - Opción **-b**: Busca solamente el archivo binario.
 - Opción **-m**: Busca solamente la página del manual.
 - Opción **-s**: Busca solamente el código fuente.
- Comando **locate**: Comando de búsqueda de archivos
 - La búsqueda se hace en una base de datos indexada (velocidad). Un archivo con la lista de todos los archivos que existen en el SO.
 - `/var/lib/mlocate/mlocate.db`
 - Generalmente, el SO ejecuta periódicamente un comando para actualizar dicha base de datos.
 - Sintaxis: `$ locate -<opciones> [patrón]`

Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **find**: Comando de búsqueda de archivos más potente.
 - Herramienta fundamental de administración. Permite buscar bajo ciertas condiciones y ejecutar acciones sobre los resultados.
 - Sintaxis: `$ find <pto_partida> -<criterio_búsqueda> -<accion_en_resultado>`
 - Criterios de búsqueda:
 - Criterio **-atime n**: Buscar ficheros abiertos hace n días (+n: hace más de n días).
 - Criterio **-mtime n**: ficheros modificados hace n días (+n ...)
 - Criterio **-newer file**: ficheros modificados después del fichero file.
 - Criterio **-size n**: con tamaño de n bloques (bloque= 512 bytes) (+n...)
 - Criterio **-type c**: tipo de fichero (f=texto, d=directorio, etc.)
 - Criterio **-fstype type**: ficheros tipo *.type
 - Criterio **-name nam**: con nombre nam
 - Criterio **-perm p**: con permisos p
 - Criterio **-user usr**: con propietario usr
 - Criterio **-nouser/nogroup**: con propietario sin entradas en /etc/passwd y etc/passwd

Sistema de Ficheros (Comandos)

Manipulación de Ficheros

- Comando **find**: (Continuación).
 - Los criterios de búsqueda se pueden combinar
 - Para forzar la precedencia: `\(... \)`
 - Condición AND: `-atime +60 -mtime +120`
 - Condición OR: `-atime +7 -o -mtime +120`
 - Condición NOT: `! -name gold.dat`
 - Acciones sobre ficheros encontrados:
 - Acción **-print**: muestra por pantalla los ficheros que concuerden.
 - Acción **-ls**: los muestra en formato extendido.
 - Acción **-exec cmd\;**: Ejecuta el comando sobre los ficheros (sin preguntar nada)
 - Acción **-ok cmd\;**: En este caso si que pregunta antes de hacerlo.
 - Acción **-xdev**: restringe la búsqueda al sistema de ficheros en el que nos encontramos.
 - Algunos ejemplos, ¿Qué hacen?
 - `$ find /home -size +2048 \(-mtime +30 -o -atime +120 \) -exec ls {} \;`
 - `$ find /home -fstype f -name core -exec rm -f {} \;`
 - `$ find /home/pepito -name '*.c' -exec mv {} /home/pepito/src \;`

Contenido de Ficheros (Comandos)

- Comando **cat**: Muestra el contenido de un fichero en un solo paso
 - Poco práctico con ficheros de gran tamaño.
- Comando **more**: Muestra el contenido de forma progresiva (paginado)
 - El número de líneas de paginado es igual al tamaño del terminal.
- Comando **less**: Evolución del comando more
 - Se trata de un programa interactivo, por lo que además de opciones, posee comandos (lanzados a través de teclas ó combinaciones de teclas).
 - **Barra Espaciadora**: Avanza un número de líneas igual al tamaño del terminal.
 - **Cursor**: Avanza/Retrocede las líneas de una en una (con **Enter** también avanza una línea).
 - **G/g**: ir al final/inicio del texto
 - **/pattern**: introducir una palabra a ser buscada avanzando en el texto.
 - **?pattern**: la palabra se busca retrocediendo en el texto.
 - **n/N**: buscar la siguiente/previa ocurrencia de la búsqueda de una palabra.
 - **AvPag/RePag**: Avanzar/retroceder una pantalla.
 - **q**: salir del programa.

Contenido de Ficheros (Comandos)

- Comando **wc**: Contar palabras de un fichero
 - Sintaxis: \$ wc -<opts> [archivo...]
 - Opción **-c**: Contar bytes.
 - Opción **-l**: Contar líneas.
 - Opción **-w**: Contar palabras.
- Comando **head**: Escribe por salida estándar la primera parte de un archivo.
 - Sintaxis: \$ head -<opciones> [archivo]
 - Opción **-c N**: Escribe los primeros N bytes.
 - Opción **-n N**: Escribe las primeras N líneas (en vez de 10, que es el valor por defecto).
- Comando **tail**: Escribe por salida estándar la última parte de un archivo.
 - Sintaxis: \$ tail -<opciones> [archivo]
 - Opciones **-c** y **-n**: Funcionan igual que en el comando head.
 - Opción **-f**: Escribe la última parte del archivo a medida que va creciendo. Muy útil para monitorear archivos de registro que van creciendo con el tiempo.

Contenido de Ficheros (Comandos)

- Comando **grep**: Escribe aquellas líneas del archivo coincidentes con un patrón.
 - Sintaxis: `$ grep -<opts> PATRON [archivos...]`
 - Opción **-c**: Imprime el número de líneas coincidentes en vez de las propias líneas.
 - Opción **-H**: Imprimir el nombre del archivo con cada coincidencia.
 - Opción **-r**: Buscar recursivamente en los subdirectorios del directorio actual.
 - Cuando el patrón contiene caracteres “especiales” (espacio, -, etc.), se puede usar entrecomillado.
 - También se pueden utilizar expresiones regulares.
 - Ejemplo: buscar líneas con palabras que empiecen por a: `grep "\ba" archivo`.
- Comando **tar**: Agrega el contenido de un árbol de directorios en un solo fichero
 - No comprime, solo empaqueta
 - Archivar: `$ tar -cvf fichero.tar /camino_a_archivar/`
 - Desarchivar: `$ tar -xvf fichero.tar`
 - Conjuntamente con **gzip** (compresor): `$ tar -czvf fichero.tar.gz /camino/`

Contenido de Ficheros (Comandos)

- Comando **cut**: Borra secciones de cada línea de un fichero.
 - Sintaxis: `$ cut -<opts> [archivos...]`
 - Opción **-c N**: Selecciona el carácter N de cada línea (-N: de inicio de línea a N)
 - Opción **-b N**: Selecciona el byte N de cada línea (M-N: del byte M al N)
 - Opción **-f N**: Selecciona el campo N. Delimitador por defecto: TAB
- Comando **sort**: Ordena las líneas de un fichero de texto
 - Sintaxis: `$ sort -<opts> [archivo]`
 - Opción **-d**: orden alfabético
 - Opción **-n**: orden numérico
 - Opción **-b**: ignorar espacios al comienzo de línea.

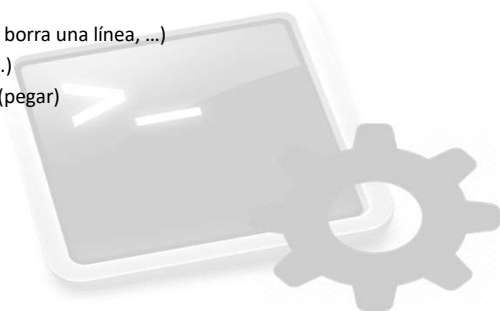
Contenido de Ficheros (Comandos)

- Comando **vi**: Editor de textos (en terminal) incluido en todo sistema UNIX.
 - Un poco desagradable de usar al principio. Con el tiempo, mucho más rápido que cualquier editor gráfico (En ocasiones será la única opción).
 - Existen versiones mejoradas como **vim** que resultan un poco más amigables.
 - Modo **comando**: salir, guardar, copiar, buscar, etc.
 - Modo **edición**: inserción de texto.
 - comando → edición: [i], [a], [o], [O],...
 - edición → comando: [Esc]
 - **Movimiento por el texto**:
 - [h],[l],[j],[k]: cursor; izda, drcha, abajo, arriba (en vim funcionan los cursores...)
 - [G]: ir a última línea ([5G]: ir a la línea 5)
 - [O][\$]: ir al inicio(cero)/final de la línea
 - Entrada a modo de edición:
 - [a][i]: añadir (append) o insertar (insert)
 - [o][O]:



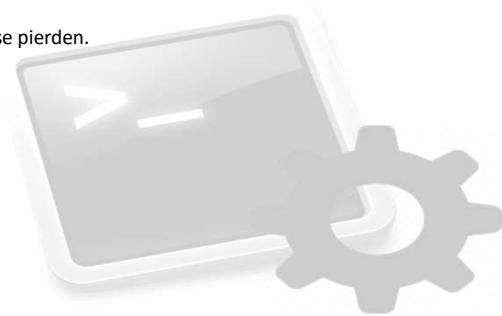
Contenido de Ficheros (Comandos)

- Comando **vi**: (Continuación)
 - **Entrada a modo de edición**:
 - [a][i]: añadir (append) o insertar (insert)
 - [o][O]: abrir debajo/encima una línea.
 - **Edición (manejo del buffer)**:
 - [x]: borra un carácter
 - [d]: borra poniendo en el buffer (cortar) ([dd] borra una línea, ...)
 - [y]: copiar dejando en el buffer (copiar) ([yy]...)
 - [p]: poner el contenido del buffer en el texto (pegar)
 - [r]: reemplazar un carácter (replace)
 - [u]: deshacer el último cambio (undo)
 - [Ctrl+r]: rehacer
 - [.] : repetir el último comando.
 - **Búsqueda**:
 - Similar a less ([/patron], [?patron], [n], [N])



Contenido de Ficheros (Comandos)

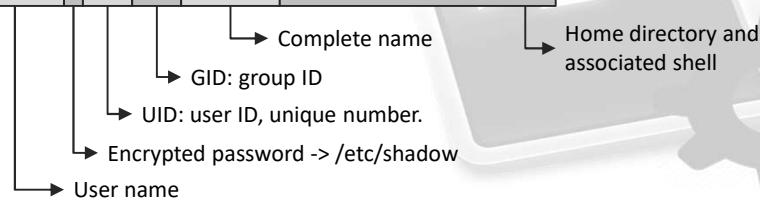
- Comando **vi**: (Continuación)
 - **Reemplazo**:
 - [%s/old/new/g]: reemplazar el string old por new en todo el texto.
 - **Salida**:
 - [:w]: guardar los cambios, sin salir.
 - [:q]: salir (falla si hay cambios sin guardar).
 - [:q!]: salida forzada, los cambios sin guardar se pierden.
 - [:wq]: guardar y salir (equivalente a [:x])



User Management

- In UNIX, users are organized in groups.
- The files `/etc/passwd` and `/etc/group` contain information about all the users and groups of the OS.
 - As well as system login, these files include basic user configuration (home directory, shell).
 - Group management: Useful to control access to certain parts of the system.
 - For each user, passwd file contains a line with the following format:

user	:	x	:	505	:	705	:	User	:	/home/usuario:/bin/bash
------	---	---	---	-----	---	-----	---	------	---	-------------------------



User Management

- The file `/etc/shadow` manages user passwords.
 - For each user, the file shadow contains a line with the following format:

```
root:$1$mFxrUn4P$0/5y9xxxBnfUXma.6hhc2..1574210:99999:7:::
```

→ Username
→ Encrypted password

← Last pass modification (days since 1 january 1970)

← Minimal number of days between pass modifications

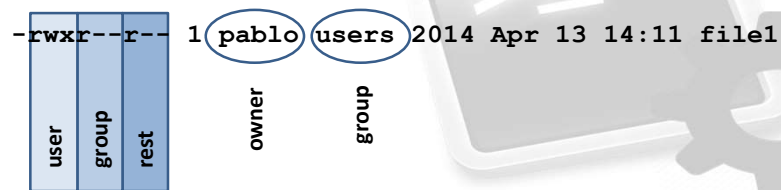
← Max number of days between pass modifications

User Management

- Based on users and groups, UNIX implements a protection mechanism for the File System based on permissions.
- Each file (and folder) has a single owner and access permissions.
- The different permissions are:
 - Read (r): allows read access to the content (list directory files)
 - Write (w): allows content modification (create/remove/move files)
 - Execute (x): execute a file (no specific extension is required (windows exe))
- File permissions can be configured according to three types:
 - User: file owner
 - Group: rest of the users from the same group as the owner
 - Rest: rest of system users

User Management

- Conventional users only have write permissions in their \$HOME directory: /home/<usuario>
 - Also in temporary directories (such as /tmp).
- Superusers (system administrators) have unlimited access to the whole file system (Warning!!)
- Information about file/directory permissions with [ls -l]:



The diagram illustrates the components of a file permission string. On the left, a table shows the permission bits for 'user', 'group', and 'rest'. In the center, the output of the 'ls -l' command is shown with 'pablo' circled as the owner and 'users' circled as the group. To the right, a terminal window icon and a gear icon are shown.

user	group	rest
-rwxr--r--		

1 pablo users 2014 Apr 13 14:11 file1

owner group

Gestión Usuarios (Comandos)

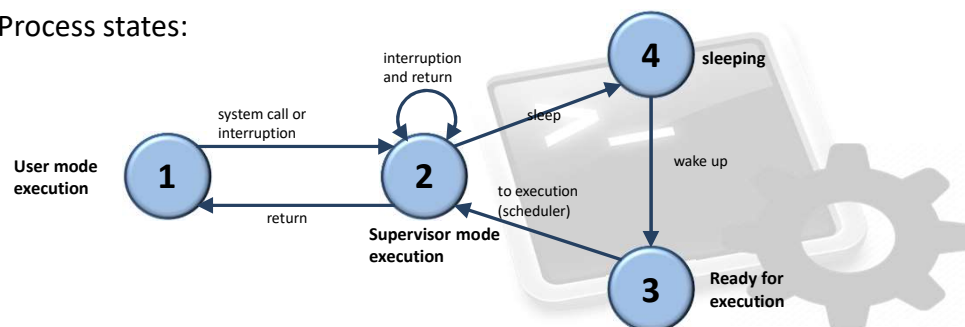
- Comando **whoami**: presenta en pantalla el nombre del usuario que lo ejecuta
- Comando **who**: muestra los usuarios conectados al sistema en ese momento
- Comando **passwd**: se utiliza para cambiar la contraseña del usuario
 - Sintaxis: \$ passwd [usuario]
 - Si no se especifica nombre de usuario se cambia el del actual.
 - Mecánica: Introducir passwd antiguo → Introducir passwd nuevo → repetir passwd nuevo
- Comando **finger**: muestra el estado de un usuario en un sistema.
 - Sintaxis: \$ finger usuario@maquina
 - Muestra info del usuario , tiempo de sesión, tiempo de inactividad, correo, fichero .plan
- Comando **write**: envía mensaje de texto a la terminal de un usuario conectado
 - Sintaxis: \$ write user [tty]
 - Comando complementario **wall**: write to all (a todos los usuarios conectados)
 - Comando **talk usuario@maquina**: establece comunicación completa (~IRC)

Gestión del Usuario (Comandos)

- Comando **chmod**: modifica los permisos de un fichero o directorio.
 - Sintaxis: `$ chmod [ugo] [+-] [rwx] [fichero o directorio]`
 - Opción **-R**: recursivo
 - Ejemplo: limitar el acceso a mi \$HOME a todos los usuarios:
 - `$ chmod -R g-rwx, o-rwx $HOME`
 - Los permisos pueden ser codificados en octal/binario: `chmod -R 700 $HOME`
- Comando **chown/chgrp**: modifica el UID/GID de un fichero.
 - Sintaxis: `$ chown [-R] nuevo_usuario fichero`
- Comando **umask**: altera los permisos asignados por defecto a nuevos ficheros.
 - Sintaxis: `$ umask [inhibition code]`
 - Establece qué bits se pondrán a 0 al crear el fichero (x siempre a 0 en ficheros).
 - Ejemplo: `$umask 022` -> los ficheros creados poseerán permisos 644 (rw-r--r--)
- Para ejecutar un fichero, permiso de ejecución activado (x).
Extensiones (.exe) NO necesarias.

Process Management

- **Process**: sequence of instructions and data stored in memory able to perform some specific task.
- Unique ID (numerical) in the system: **PID**
- Three main memory segments: code/data/stack
- Process states:



Process Management

- Processes have a hierarchy similar to the file system (tree). Root process: **init**
 - Each process (excluding init) has a father process.
 - The kernel (root) has absolute control of every system process.
- A process can be identified by its PID.
 - Only its owner can interact with that process (UID).
- The shell is a process, dependent of the terminal.
 - Foreground process: blocks shell utilization until it finishes execution
 - \$ ls -R / >/dev/null
 - Background process: does not block shell
 - \$ ls -R / >/dev/null &
 - Processes can be moved between foreground and background:
 - [Ctrl+z]: foreground process stopped (suspend execution)
 - bg** move process to background mode and **fg** moves it back to foreground.

Process Management

- /proc: pseudo file system associated to the processes.
 - Employed as interface to the data structures in the kernel associated to each process.
 - Content example(one folder for each process):

```
[ root si /tmp ] ls /proc/
1      211  2428  2490  2600  41    7    bus      execdomains  kalleysms  misc          scsi          timer_stats
1076   212  2439  2497  2603  42    741   cgroups    fb           kcore       modules       self          tty
1153   213  2440  2512  2605  4769  742   cmdline   filesystems  key-users   mounts        slabinfo     uptime
1620   2318  2459  2521  2618  4772  774   cpuinfo    fs           kmsg        mpt           stat         version
1687   2329  2465  2532  2691  5     775   crypto     ide          kpagecount  mtrr         swaps        vmallocinfo
173    2339  2468  2566  2719  5280  958   devices    interrupts  kpageflags  net          sys          vmlstat
2      2397  2470  2594  3     5282  acpi       diskstats   iomem       loadavg      pagetypeinfo sysrq-trigger zoneinfo
2099   2410  2483  2596  39    5387  asound     dma          ioports     locks        partitions   sysvipc
210    2420  2489  2598  4     6     buddyinfo  driver       irq          meminfo       sched_debug  timer_list
```

— In each folder...

```
[ root si /tmp ] ls /proc/2719
attr      clear_refs  cpuset      exe      io      maps      mounts      oom_adj  root      smaps      status
auxv      cmdline     cwd         fd       limits  mem       mountstats  oom_score sched     stat      task
cgroup    coredump_filter environ      fdinfo   loginuid mountinfo  net       sessionid statm     tchan
```

- fd**: files opened by the process.
- maps**: Physical memory range associated to the process.
- stat**: current process status: PID, PPID, utime, etc.

Gestión de Procesos

- Comando **top**: monitorización de procesos en tiempo real.
 - En ejecución, podemos utilizar comandos interactivos:
 - **k**: matar un proceso indicando su PID.
 - **u**: mostrar solo los procesos de un usuario específico.
- Comando **ps**: reporta información sobre procesos activos.
 - Sintaxis: `ps -<opciones>`
 - Opción **-e**: muestra todos los procesos.
 - Opción **-u user**: muestra los procesos de un usuario concreto
 - Opción **-f**: full-format listing.
- Comando **kill**: envío de señales a un proceso
 - Sintaxis: `kill -<opciones> <pid>`
 - Opción **-s**: Cambia el tipo de señal a enviar (`-s 9 = -SIGKILL = -9`)
- Comando **pstree**: relaciones de herencia entre procesos.

