

Sesión 3: `man` `bash`

Programación para Sistemas

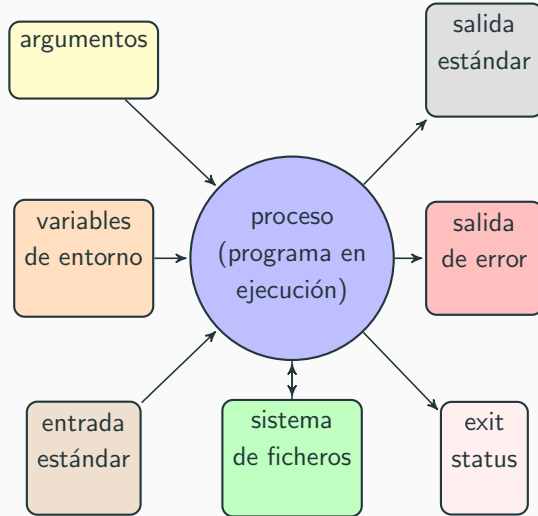
Ángel Herranz

Curso 2024-2025

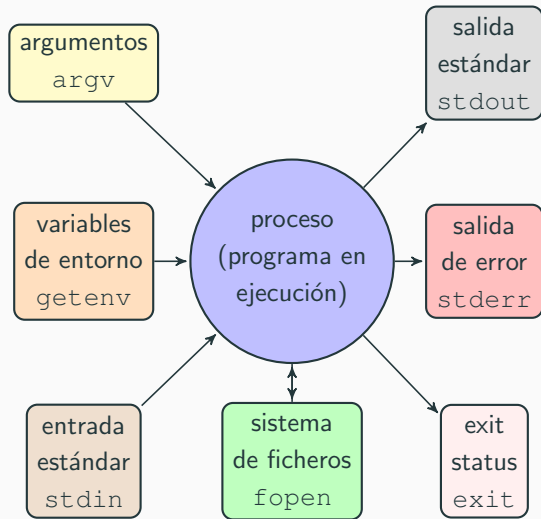
Recordatorio i

- *Todo* son ficheros y procesos en Unix
- Ficheros: nombres, permisos, propietarios
- Nombrado de ficheros
 - `http://refspecs.linuxfoundation.org/fhs.shtml`
 - Absoluto, ej. `/etc/password`
 - Relativo, ej. `../../etc/password` (desde es `/home/angel`) o `./secuencia`
- Línea de comandos: mandatos (ejecutando programas)
- Configuración: ficheros de configuración y variables de entorno

Recordatorio ii



Recordatorio ii

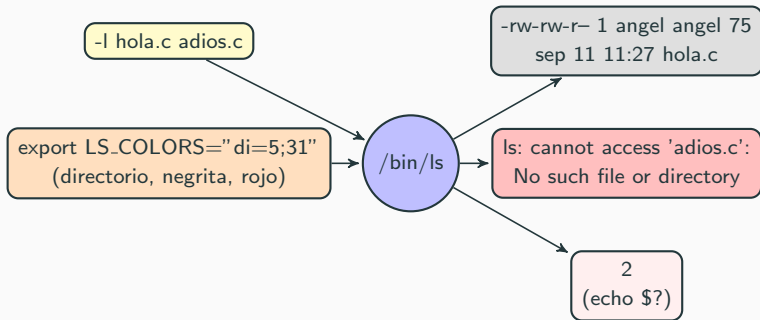


Ejemplo i

```
ls -l hola.c adios.c
```

Ejemplo i

```
ls -l hola.c adios.c
```

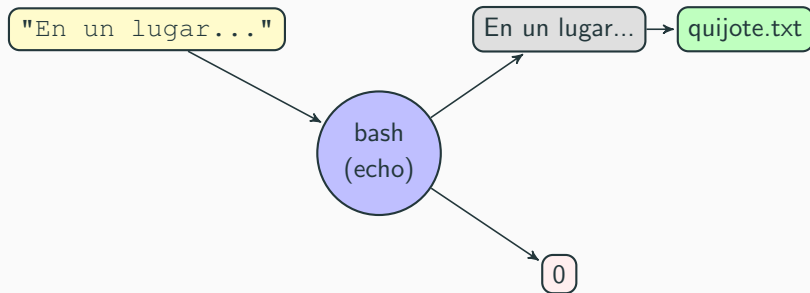


Ejemplo ii

```
echo "En un lugar de la mancha..." > quijote.txt
```

Ejemplo ii

```
echo "En un lugar de la mancha..." > quijote.txt
```

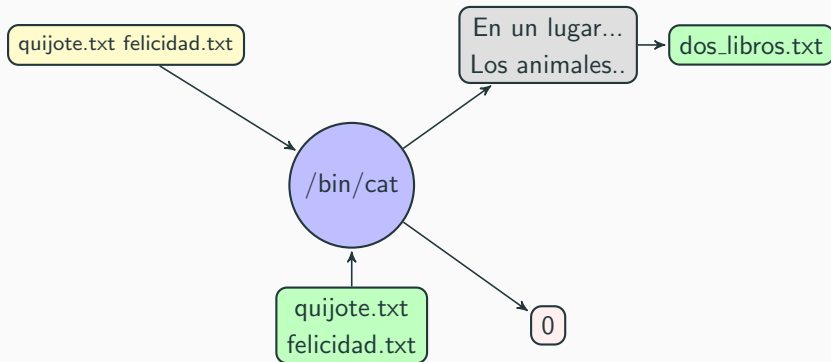


Ejemplo iii

```
cat quijote.txt felicidad.txt > dos_libros.txt
```

Ejemplo iii

```
cat quijote.txt felicidad.txt > dos_libros.txt
```



man bash

y algún programa útil como **test**

- Expansión,
- redirección,
- control de flujo, y
- operadores de control.

El juego de la expansión

- “No es oro todo lo que reluce...”
- Bash se basa en el concepto de expansión:

Antes de interpretar la línea de comandos
primero se expanden ciertas expresiones

- Expansión de variables: **echo** \$EDAD $\rightsquigarrow$ **echo** 23
- Es como haber escrito directamente **echo** 23
- Otras expansiones: $\sim$ o $*$

man bash

```
BASH(1)                                General Commands Manual                                BASH(1)
NAME
    bash - GNU Bourne-Again SHell
SYNOPSIS
    bash [options] [command_string | file]
COPYRIGHT
    ...
DESCRIPTION
    ...
OPTIONS
    ...
ARGUMENTS
    ...
INVOCATION
    ...
DEFINITIONS
    ...
RESERVED WORDS
    ...
SHELL GRAMMAR
    Simple Commands
        A simple command is a sequence of optional variable assignments followed by blank
        separated words and redirections, and terminated by a control operator. The first
        word specifies the command to be executed, and is passed as argument zero. The
        remaining words are passed as arguments to the invoked command.
    ...
```

Secciones relevantes en el manual `i`

NAME

SYNOPSIS

COPYRIGHT

DESCRIPTION

OPTIONS

ARGUMENTS

INVOCATION

DEFINITIONS

(hasta aquí `man` como con cualquier otro *programa*)

Secciones relevantes en el manual ii

RESERVED WORDS

(¡es un lenguaje de programación!)

SHELL GRAMMAR

(detalles más adelante)

Simple Commands

(ej. `ls -al`)

Pipelines

(`cmd1 | cmd2`)

Lists

(`cmd1 && cmd2 o cmd1 || cmd2`)

Compound Commands

(**for**, **if**, **case**, **while**...)

Coprocesses

(no lo vemos)

Shell Function Definitions

(lo vemos en scripts)

COMMENTS

(líneas empezando en #)

Secciones relevantes en el manual iii

PARAMETERS

Positional Parameters

(variables de entorno y parámetros)

Special Parameters

(lo vemos en scripts: \$1, \$2, ...)

(\$?, \$0, \$#, \$*, ...)

Shell Variables

(ej. MAX_OUTPUT=100)

Arrays

(no lo vemos)

Secciones relevantes en el manual iv

EXPANSION

(detalles más adelante, ¡importante!)

Brace Expansion

(ej. **echo** xxx{hola,adios}, no lo vemos)

Tilde Expansion

(ej. ~)

Parameter Expansion

(ej. \$FECHA o \${FECHA})

Command Substitution

(detalles más adelante)

Arithmetic Expansion

(no lo vemos)

Process Substitution

(no lo vemos)

Word Splitting

(no lo vemos)

Pathname Expansion

(ej. `ls -al *`)

Quote Removal

(tras la expansión \, ' y " se ignoran)

Secciones relevantes en el manual v

REDIRECTION

Redirecting Input	<code>(cmd < file)</code>
Redirecting Output	<code>(cmd > file y cmd 2> file)</code>
Appending Redirected Output	<code>(cmd >> file y cmd 2>> file)</code>
Redirecting Standard Output and Standard Error	<code>(cmd &> file)</code>
Appending Standard Output and Standard Error	<code>(cmd &>> file)</code>
Here Documents and Strings	<code>(no lo vemos)</code>
Duplicating File Descriptors	<code>(no lo vemos)</code>
Moving File Descriptors	<code>(no lo vemos)</code>
Opening File Descriptors for Reading and Writing	<code>(no lo vemos)</code>

Secciones relevantes en el manual vi

ALIASES

(ej. **alias** la=ls -al)

ARITHMETIC EVALUATION

(ej. **echo** \$((1+2)) o **echo** \$(A++))

CONDITIONAL EXPRESSIONS

(detalles más adelante)

(las siguientes secciones describen formalmente el orden en el que Bash interpreta un comando, la expansión, la redirección, etcétera, no lo vemos)

SIMPLE COMMAND EXPANSION

COMMAND EXECUTION

COMMAND EXECUTION ENVIRONMENT

ENVIRONMENT

Secciones relevantes en el manual vii

EXIT STATUS	(ya lo conocemos, detalles más adelante)
JOB CONTROL	(&, Ctrl-Z, fg , bg)
PROMPTING	(ej. PS1="# ", no lo vemos)
READLINE	(biblioteca para editar la línea, no lo vemos)

HISTORY

(Bash recuerda lo que has ejecutado)

HISTORY EXPANSION(!!, !ls, !2)

Secciones relevantes en el manual ix

SHELL BUILTIN COMMANDS

source (también **.**, **no** confundir con **./**)

alias

bg y **fg**

cd y **pwd**

echo y **read**

exec

exit

kill y **ulimit**

popd y **pushd**

...

Secciones relevantes en el manual x

RESTRICTED SHELL

(modo restringido, no lo vemos,
ej. no permite cambiar de directorio)

SEE ALSO

(desde aquí `man` como con cualquier otro *programa*)

FILES

AUTHORS

BUG REPORTS

BUGS

Expansión: parámetros (aka variables)

 `man bash` y busca la sección *EXPANSION*, en concreto *Parameter Expansion*:

Expansión: parámetros (aka variables)

-  man bash y busca la sección *EXPANSION*, en concreto *Parameter Expansion*:

`${parameter}`

- Se puede escribir sin llaves: `${A}` $\equiv$ `$A`
- *Parameter Expansion* (variantes):

`${parameter:-word}`

`${parameter:?word}`

`${parameter:offset:length}`

`${#parameter}`

etc.

-  `FECHA=2019-12-17` y entonces ¿a qué *expande* `${#FECHA}` o `${FECHA:2:2}`?

Expansión: command substitution



Guardar en FECHA el resultado de `date +%Y-%m-%d`

¹Comillas invertidas

Expansión: command substitution

 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- `man bash` y busca **Command Substitution**:

`$ (command)`

¹Comillas invertidas

Expansión: command substitution

 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- `man bash` y busca **Command Substitution**:

`$(command) ≡ `command`1`

¹Comillas invertidas

Expansión: command substitution

📄 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- `man bash` y busca **Command Substitution**:

`$(command) ≡ `command`1`

- Es una expansión **extraordinariamente útil**
- Ejemplo:

```
$ FECHA=`date +%Y-%m-%d`
```

¹Comillas invertidas

Expansión: command substitution

📄 Guardar en FECHA el resultado de `date +%Y-%m-%d`

- `man bash` y busca **Command Substitution**:

$\$(\text{command}) \equiv \text{'command'}$ ¹

- Es una expansión **extraordinariamente útil**
- Ejemplo:

```
$ FECHA=`date +%Y-%m-%d`
```

- Y luego...

```
$ echo ${FECHA}
```

```
$ echo ${FECHA:5:2}
```

¹Comillas invertidas

No es necesario aprenderse el manual

No es necesario aprenderse el manual

Pero es necesario aprender a usarlo



En el manual de Bash se puede leer la siguiente descripción sobre expansión de variables (en realidad es más compleja pero estas líneas son suficientes):

```
${!prefix*}
```

Nombres que encajan con prefijo. Expande a los nombre de variables que empiezan por prefix separados por espacios y ordenados por orden alfabético.

Se pide escribir las cuatro líneas de la salida estándar resultado de la ejecución de los siguientes mandatos Bash, suponiendo que no hay otras variables definidas que empiecen por “MIV”:

```
$ MIVUNO=
$ MIVDOS=
$ MIVTRES=
$ MIVCUATRO=
$ MIVCINCO=
$ echo ${!MIV*}
$ echo ${!MIVC*}
$ echo ${!MIVT*}
$ echo ${!MIVU*}
```

Redirección

 `man bash` y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

Redirección

 man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

```
command < file
```

```
command > file
```

```
command >> file
```

```
command 2> file
```

```
command 2>> file
```

Redirección

 man bash y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

```
command < file
```

```
command > file
```

```
command >> file
```

```
command 2> file
```

```
command 2>> file
```

- Otras redirecciones *muy* interesantes:

```
command 2>&1
```

```
command > file 2>&1  $\equiv$  command &> file
```

Redirección

-  `man bash` y busca la sección *REDIRECTION*, en concreto *Redirecting Input* y *Redirecting output*:

```
command < file
```

```
command > file
```

```
command >> file
```

```
command 2> file
```

```
command 2>> file
```

- Otras redirecciones *muy* interesantes:

```
command 2>&1
```

```
command > file 2>&1  $\equiv$  command &> file
```

- *Más*: *here documents*, *duplicating file descriptors*, etc.

Control de flujo i

- if

`if command; then command; else command; fi`

≡

`if list; then list; else list; fi`

- for

`for name in words: do list; done`

- Otras estructuras en el [manual](#):

`case, while`, etc.

Control de flujo ii

- El programa **test**: man **test**
- Comprobaciones sobre **ficheros y strings**:

SYNOPSIS

test EXPRESSION

...

EXPRESSION sets exit status. It is one of:

...

-n STRING

the length of STRING is nonzero

STRING1 = STRING2

the strings are equal

...

-e FILE

FILE exists

Control de flujo iii

 Ejecuta `which` [

- ¿Qué es `which` [?

Control de flujo iii

 Ejecuta `which` [

- ¿Qué es `which` [?

 Ejecuta

```
$ [
```

```
$ [ ]
```

```
$ [ -n "Hola"]
```

```
$ [ -z "Hola"]
```

```
$ test -n "Hola"
```

```
$ test -z "Hola"
```

- ¿Qué está pasando?

Control de flujo iii

📄 Ejecuta `which` [

- ¿Qué es `which` [?

📄 Ejecuta

```
$ [
```

```
$ [ ]
```

```
$ [ -n "Hola"]
```

```
$ [ -z "Hola"]
```

```
$ test -n "Hola"
```

```
$ test -z "Hola"
```

- ¿Qué está pasando?

```
$ if [ -e $VIDEO ]; echo "$VIDEO existe"; fi
```

Control de flujo iv

- Operadores de control

`command ; command` (también `command ;`)

`command & command` (también `command &`)

`command | command`

`command && command`

`command || command`

Control de flujo iv

- Operadores de control

`command ; command` (también `command ;`)

`command & command` (también `command &`)

`command | command`

`command && command`

`command || command`

- Observad las equivalencias que usan los programadores

```
command1 && command2
```

```
command1 || exit 1  
command2
```

```
if command1; then command2; fi
```

```
if command1; then  
    command2;  
else  
    exit 1;  
fi
```

Background y Foreground

- Cuando se ejecuta un comando Bash se espera a que el programa termine (*foreground*), por ejemplo:

```
$ gedit
```

- El operador de control & permite decirle a Bash que no espere a que el comando termine (*background*)
- Prueba:

```
$ gedit &
```

- En esto entra en juego `Ctrl-Z`, **bg** y **fg**

read

🕒 2' 🔍 ¿Qué es read?

read

🕒 2' 🔍 ¿Qué es read?

💻 Ejecutar y explorar:

```
$ read -p "¿Qué edad tienes?" EDAD
```

⚠️ La respuesta queda en la variable de entorno EDAD:

```
$ echo $EDAD
```

- Otro ejemplo

```
$ read -p "¿Entorno habla?" VARHABLA; export VARHABLA; ./habla
```

- ¿Crawling?

²Instalado en triqui, instalable en Ubuntu con `apt-get install curl`

Crawling i

- ¿Crawling?
- Usaremos el programa **cURL**²: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

²Instalado en `triqui`, instalable en Ubuntu con `apt-get install curl`

Crawling i

- ¿Crawling?
- Usaremos el programa **cURL**²: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

- ¿Qué es grep?

²Instalado en `triqui`, instalable en Ubuntu con `apt-get install curl`

Crawling i

- ¿Crawling?
- Usaremos el programa **cURL**²: `man curl`

```
$ curl http://www.fi.upm.es
```

```
$ curl http://www.fi.upm.es | grep href
```

- ¿Qué es grep?
- *Crawling*:

```
$ curl -s http://www.fi.upm.es | grep -Po '(?<=href=")[^"]*(?=")'
```

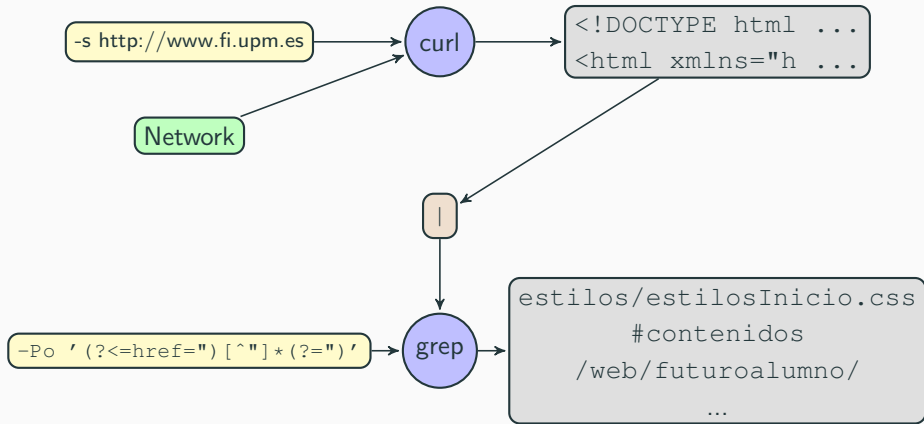
²Instalado en triqui, instalable en Ubuntu con `apt-get install curl`

Crawling ii

```
curl -s http://www.fi.upm.es | grep -Po '(?<=href=")[^"]*(?=")'
```

Crawling ii

```
curl -s http://www.fi.upm.es | grep -Po '(?<=href=")[^"]*(?=")'
```



- Casi *mágico*:

```
$ curl -s http://www.fi.upm.es |  
  grep -Po '(?<=href=") [^"]* (?=")' |  
  sort |  
  uniq
```