

# Sesión 09: Bash y Conceptos UNIX

Programación para Sistemas

---

Ángel Herranz

2022-2023

Universidad Politécnica de Madrid

## Sesión 0: *Bash Crash Course*

## Sesión 0: *Bash Crash Course*

★ <https://guide.bash.academy/>  
¿Quién hizo esta guía?

## Cuaderno *UNIX, Shell y Scripts*

*Francisco Rosales*  
*Ángel Herranz*

(para todas las sesiones de Bash  
disponible en la Web de Herranz)

## En el capítulo de hoy...

- ¿Qué es Bash?
- **Conceptos** de sistemas operativos:
  - Ficheros, programas, procesos, variables de entorno, entrada/salida, invocación, estado de terminación, etc.
- **Sintaxis** en la línea de comandos
- Empezamos con el juego de la **expansión**

Durante las siguientes semanas, **muy recomendable**:

# En el capítulo de hoy...

- ¿Qué es Bash?
- **Conceptos** de sistemas operativos:
  - Ficheros, programas, procesos, variables de entorno, entrada/salida, invocación, estado de terminación, etc.
- **Sintaxis** en la línea de comandos
- Empezamos con el juego de la **expansión**

Durante las siguientes semanas, **muy recomendable**:

🏠 Repasar sesión 0

🏠 “Cuaderno de UNIX”

🏠 <https://guide.bash.academy/>

🏠 *The UNIX Programming Environment*  
Kernighan y Pike

# Objetivo: comunicarse con los procesos i

 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

# Objetivo: comunicarse con los procesos i

💬 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

línea de comandos

💬 ¿Cómo podemos *enviar* información a nuestro programa?

# Objetivo: comunicarse con los procesos i

💬 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

línea de comandos

💬 ¿Cómo podemos *enviar* información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: `stdin` (ficheros en general)

# Objetivo: comunicarse con los procesos i

💬 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

línea de comandos

💬 ¿Cómo podemos *enviar* información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: `stdin` (ficheros en general)
- *Variables de entorno* ⚠️

# Objetivo: comunicarse con los procesos i

¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

## línea de comandos

¿Cómo podemos *enviar* información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: `stdin` (ficheros en general)
- *Variables de entorno* ⚠

¿Cómo podemos *recibir* información de nuestro programa?

# Objetivo: comunicarse con los procesos i

💬 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

## línea de comandos

💬 ¿Cómo podemos *enviar* información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: `stdin` (ficheros en general)
- *Variables de entorno* ⚠️

💬 ¿Cómo podemos *recibir* información de nuestro programa?

- Salida estándar: `stdout` (ficheros en general)
- Y salida de error: `stderr`

# Objetivo: comunicarse con los procesos i

💬 ¿Cómo ponemos en marcha nuestros programas (en ejecución = proceso)?

## línea de comandos

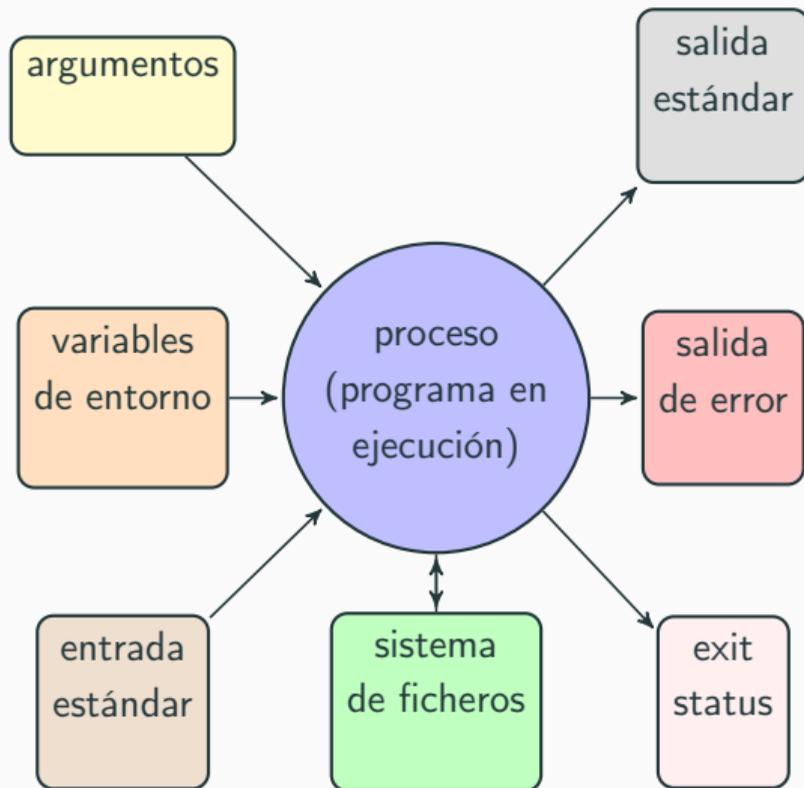
💬 ¿Cómo podemos *enviar* información a nuestro programa?

- Parámetros en la línea de comandos
- Entrada estándar: `stdin` (ficheros en general)
- *Variables de entorno* ⚠️

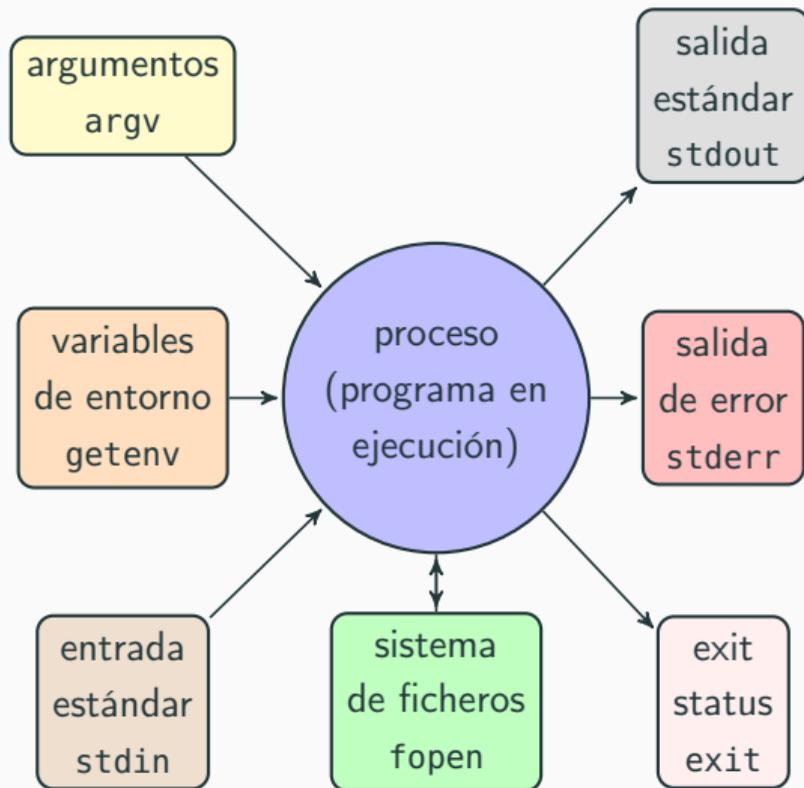
💬 ¿Cómo podemos *recibir* información de nuestro programa?

- Salida estándar: `stdout` (ficheros en general)
- Y salida de error: `stderr`
- Estado de terminación: `exit` o `return en main`

# Objetivo: comunicarse con los procesos ii



# Objetivo: comunicarse con los procesos ii



## ¿Otras formas que no vemos hoy?

- Ficheros además de `stdin`, `stdout` o `stderr` (ya se han usado en ejercicios propuestos)
- *Pipes* (pipe, se manejan como ficheros)
- *Signals*: `kill` (ejecutar `kill -l`)
- *Sockets* (socket)
- Memoria compartida (`mmap`, basado en fichero)

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

```
$ ls -al / ~
```

## Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

## Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

```
$ ls      -al  /    ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```

 Sin espacios alrededor de =

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```

 Sin espacios alrededor de =

Equivalente a

```
$ export MAX_OUTPUT=100
```

```
$ ./secuencia -30 0 -3
```

```
$ unset MAX_OUTPUT
```

# Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [arg3 [...]]]]
```

```
$ ls -al / ~
```

```
$ variable=valor mandato [arg1 [arg2 [...]]]
```

```
$ MAX_OUTPUT=100 ./secuencia -30 0 -3
```

 Sin espacios alrededor de =

Equivalente a

Y luego...

```
$ export MAX_OUTPUT=100
```

```
#include <stdlib.h>
```

```
$ ./secuencia -30 0 -3
```

```
...
```

```
$ unset MAX_OUTPUT
```

```
char *limit = getenv("MAX_OUTPUT");
```

🕒 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 "¿Límite?" MAX_OUTPUT; export MAX_OUTPUT; ./secuencia 0 100 1
```

# El interés del *exit status*

- 📄 Escribir un programa que termine *mal* (`mal.c`)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

# El interés del *exit status*

- 📄 Escribir un programa que termine *mal* (*mal.c*)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

```
$ ./mal
```

```
$ echo $?
```

```
255
```

```
$ echo $?
```

```
0
```

- Y esto, *¿Para qué sirve?*

# El interés del *exit status*

- 📄 Escribir un programa que termine *mal* (mal.c)
- 📄 Ejecutar y comprobar *cómo de mal ha terminado*

```
$ ./mal
```

```
$ echo $?
```

```
255
```

```
$ echo $?
```

```
0
```

- Y esto, *¿Para qué sirve?*

```
$ if ./mal; then echo BIEN; else echo MAL; fi
```

```
MAL
```



¡Atención a la sintaxis del **if**!

# Explorar el sistema de ficheros i

- En UNIX no hay *unidades de disco* (C:, D:, ...)
- Todo empieza en el directorio raíz: /

# Explorar el sistema de ficheros i

- En UNIX no hay *unidades de disco* (C:, D:, ...)

- Todo empieza en el directorio raíz: /

## Nombrado de ficheros y directorios

- **Absoluto**: el nombre empieza por /

Ej. /etc/password, /home/angel/Asignaturas/pps.txt

- **Relativo**: el nombre **no** empieza por / y se convierte en absoluto concatenándolo al *working directory*.

Ej. ../../etc/password, Asignaturas/pps.txt

- Elementos especiales: ~, ., ...

Ej. ../../etc/password, ~/Asignaturas/pps.txt, ./secuencia

¡Busca la **equivalencia** con las **carpetas**!

- Empezar en / (**cd /**)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`
- Moverse a ~ (**cd ~**)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`

¡Busca la **equivalencia** con las **carpetas**!

- Empezar en / (**cd /**)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`
- Moverse a ~ (**cd ~**)
- Ejecutar `ls -al`
- Entrar en cada directorio y ejecutar `ls -al`

 5'  <http://refspecs.linuxfoundation.org/fhs.shtml>

❓ ¿Dónde están los discos?

### ? ¿Dónde están los discos?

- Los discos están en `/dev` (ej. `/dev/sda`, `/dev/sdb1`, `/dev/hda`, `/dev/hda1`, etc.)

- Pero **no se pueden usar directamente**

- Primero hay que *montarlos*:

```
$ mount /dev/sda1 /home
```

```
$
```

- Y entonces el directorio `/home` **es realmente** la partición 1 del disco `/dev/sda`

## Explorar el sistema de ficheros iii

❓ ¿Dónde están los discos?

- Los discos están en `/dev` (ej. `/dev/sda`, `/dev/sdb1`, `/dev/hda`, `/dev/hda1`, etc.)
- Pero **no se pueden usar directamente**
- Primero hay que *montarlos*:

```
$ mount /dev/sda1 /home  
$
```

- Y entonces el directorio `/home` **es realmente** la partición 1 del disco `/dev/sda`

🔍 `cat /etc/fstab`

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué es ls?

which ls

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué es `ls`?

```
which ls
```

- ¿Qué hace `ls`?

```
man ls
```

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué es `ls`?

`which ls`

- ¿Qué hace `ls`?

`man ls`

- ¿Y `cd`?

`which cd? man cd?`

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué es `ls`?

`which ls`

- ¿Qué hace `ls`?

`man ls`

- ¿Y `cd`?

`which cd?` `man cd?`

 ¿Por qué no hay programa ni manual de `cd`?

# ¿Qué hay detrás de los mandatos conocidos?

- ¿Qué es `ls`?

`which ls`

- ¿Qué hace `ls`?

`man ls`

- ¿Y `cd`?

`which cd?` `man cd?`

 ¿Por qué no hay programa ni manual de `cd`?

- `cd` es un *built in command* de Bash:

**man bash**

y buscar "SHELL BUILTIN COMMANDS"

- Los mandatos de Bash pueden ser

programas

o

*built in commands*

- Explorar el manual: man bash y man PROGRAMA (donde PROGRAMA es tu programa favorito como **cat**, grep, etc.)

# ¿Dónde están los programas?

- Los programas están en el sistema de ficheros

 `which ls`

 ¿Cómo busca? ¿Y si hay dos programas con el mismo nombre?

Variable de entorno **PATH**

# ¿Dónde están los programas?

- Los programas están en el sistema de ficheros

 `which ls`

 ¿Cómo busca? ¿Y si hay dos programas con el mismo nombre?

## Variable de entorno **PATH**

- Un **path** es una lista de directorios<sup>1</sup>

 Mira y cambia el PATH

```
$ echo $PATH
```

```
$ ls
```

```
$ which ls
```

```
$ PATH=
```

```
$ echo $PATH
```

```
$ ls
```

```
$ which ls
```

---

<sup>1</sup>Herranz Separadas por el caracter ":" en Unix

# Sobre las variables de entorno i

- Los programas usan variables de entorno, algunas son **comunes**:

PATH, PS1, USER, SHELL, PWD, HOSTNAME, LANG,  
EDITOR, etc.

- Pero cada programador puede **definir** las suyas:

JAVA\_HOME, CLASSPATH, MAX\_OUTPUT

- Todo lo que se haga con ellas **se pierde** entre sesiones

# Sobre las variables de entorno

- Se establecen al arrancar Bash aprovechando los **ficheros de inicialización**
- **/etc/profile** *The systemwide initialization file, executed for login shells*
- **/etc/bash.bashrc** *The systemwide per-interactive-shell startup file*
- **~/.bash\_profile** *The personal initialization file, executed for login shells*
- **~/.bashrc** *The individual per-interactive-shell startup file*
- **~/.bash\_logout** *The individual login shell cleanup file, executed when a login shell exits*
- **/etc/bash.bash.logout** *The systemwide login shell cleanup file, executed when a login shell exits*

🕒 1' 🔍 man **echo**

🕒 1' 🔍 man **echo**

📄 Ejecutar y *diseccionar*

```
$ echo Fíjate en los     espacios
```

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

🕒 1' 🔍 man **echo**

📄 Ejecutar y *diseccionar*

```
$ echo Fíjate en los     espacios
```

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

🏠 ¿Te atreves a programar echo en C?

🕒 1' 🔍 man **echo**

📄 Ejecutar y *diseccionar*

```
$ echo Fíjate en los espacios
```

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

🏠 ¿Te atreves a programar echo en C?

💬 ¿Qué es echo? ¿Dónde está? ¿Por qué aparece en negrita en estas transparencias? ¿Has probado which? ¿Has mirado en man bash?

 Descargar El Quijote en texto plano

[https://babel.upm.es/~angel/teaching/pps/don\\_quijote.txt](https://babel.upm.es/~angel/teaching/pps/don_quijote.txt)

 Descargar El Quijote en texto plano

[https://babel.upm.es/~angel/teaching/pps/don\\_quijote.txt](https://babel.upm.es/~angel/teaching/pps/don_quijote.txt)

- ¿Qué hace cat?

 1'  man **cat**

 Descargar El Quijote en texto plano

[https://babel.upm.es/~angel/teaching/pps/don\\_quijote.txt](https://babel.upm.es/~angel/teaching/pps/don_quijote.txt)

- ¿Qué hace cat?

 1'  man **cat**

 Ejecutar y *diseccionar*

```
$ cat quijote.txt
```

```
$ cat
```

 ¿Qué ocurre?

 Descargar El Quijote en texto plano

[https://babel.upm.es/~angel/teaching/pps/don\\_quijote.txt](https://babel.upm.es/~angel/teaching/pps/don_quijote.txt)

- ¿Qué hace cat?

 1'  man **cat**

 Ejecutar y *diseccionar*

```
$ cat quijote.txt
```

```
$ cat
```

 ¿Qué ocurre? Prueba a escribir

Los animales son felices mientras  
tengan salud y suficiente comida.

**Ctrl-d**

## Ejecutar y *diseccionar*

```
$ cat < quijote.txt
```

```
$ cat > la_conquista.txt
```

```
Los animales son felices mientras  
tengan salud y suficiente comida.
```

```
Ctrl-d
```

```
$ cat quijote.txt la_conquista.txt
```

```
$ cat quijote.txt la_conquista.txt > dos_libros.txt
```

# Conectando salida estándar y entrada estándar

🕒 2' 🔍 man grep, man wc

📖 Jugamos con El Quijote:

- ¿Cuántas líneas tiene el fichero descargado?
- Buscar líneas con “Sancho”
- Contar número de líneas con “Sáncho”