Sesión 11: En línea de comandos

Programación para Sistemas

Ángel Herranz

2020-2021

Universidad Politécnica de Madrid

Cuaderno

UNIX, Shell y Scripts

Fracisco Rosales

Ángel Herranz

En el capítulo de hoy...

- ¿Cómo nos comunicamos con nuestros programas?
- Mandatos sencillos
- Un poquito de redirección
- Variables de entorno
- Comandos últiles

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  línea de comandos

- ¿Cómo podemos enviar información a nuestro programa?

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  línea de comandos

- ¿Cómo podemos enviar información a nuestro programa?
  - Parámetros en la línea de comandos
  - Entrada estándar: `stdin` (ficheros en general)

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  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 ⚠

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  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?

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  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)
  - Salida de error: `stderr`

[1] programa en ejecución = proceso

¿Cómo nos comunicamos con los procesos?

- ¿Cómo ponemos en marcha nuestros programas[1]?:

  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)
  - Salida de error: `stderr`
  - Estado de terminación: **`exit`** o **`return`** en main

[1] programa en ejecución = proceso

¿Otras formas que no vemos hoy?

- Ficheros además de `stdin`, `stdout` o `stderr`
- Ya se han usado en la práctica
- *Pipes*
- *Signals*: **`kill`** (ejecutar **`kill -l`**)
- *Sockets*
- Memoria compartida

Mandatos, argumentos y variables de entorno

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

Mandatos, argumentos y variables de entorno

```
$ mandato [arg1 [arg2 [...]]]
$ ls -al / ~
```

Mandatos, argumentos y variables de entorno

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

$ ls -al / ~

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

Mandatos, argumentos y variables de entorno

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

Y luego...

```
#include <stdlib.c>
...
char *limit =
  getenv("MAX_OUTPUT");
```

secuencia.c

- Escribir un programa `secuencia.c` que lea tres argumentos `FIRST`, `LAST`, `INCR` e imprima en la salida estándar un número por línea empezando en `FIRST`, incrementando en `INCR` sin pasar de `LAST`. Si la variable de entorno `MAX_OUTPUT` está definida entonces no deberán imprimirse más de las líneas indicadas en dicha variable
- 5' Primera iteración: imprimir el valor de `MAX_OUTPUT`
- Terminar el programa (ten en cuenta que `FIRST` puede ser mayor que `LAST` e `INCR` puede ser negativo)

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

Un uso más interesante del *exit status*

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

Un uso más interesante 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?

Un uso más interesante 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 raiz: `/`

Explorar el sistema de ficheros i

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

🔔 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`

Explorar el sistema de ficheros ii

¡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`

Explorar el sistema de ficheros ii

¡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`

Explorar el sistema de ficheros iii

- ¿Dónde están los discos?

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`

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 hace `ls`?

```
man ls
```

¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

  `man ls`

- ¿Qué hace `cd`?

  `man` **`cd`**

¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

  `man ls`

- ¿Qué hace `cd`?

  `man` **`cd`**

👎 ¿Por qué no hay manual de **`cd`**?

¿Qué hay detrás de los mandatos conocidos?

- ¿Qué hace `ls`?

  `man ls`

- ¿Qué hace `cd`?

  `man` **`cd`**

👎 ¿Por qué no hay manual de **`cd`**?

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

`man ls`

man cd

- Los mandatos de Bash pueden ser

  programas

  o

  built in commands

- Explorar el manual: `man bash` y `man PROGRAMA`

¿Dónde están los programas?

- Los programas están en el sistema de ficheros

 which ls

 ¿Y si hay dos programas con el mismo nombre?

Variable de entorno PATH

[2] Separadas por el caracter ":" en Unix

¿Dónde están los programas?

- Los programas están en el sistema de ficheros

which ls

¿Y si hay dos programas con el mismo nombre?

Variable de entorno PATH

- Un path es una lista de directorios[2]

Mira y cambia el PATH

```
$ echo $PATH
$ ls
$ which ls
```

```
$ PATH=
$ echo $PATH
$ ls
$ which ls
```

[2]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`

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

Sobre las variables de entorno

- Se establecen al arrancar Bash
- `/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*

echo

1' `man` **echo**

echo

1' man **echo**

Ejecutar y *diseccionar*

```
$ echo Fíjate en los    espacios
$ echo "En un lugar de la mancha..." > quijote.txt
```

echo

- 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?

echo

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`?

cat i

- Descargar El Quijote en texto plano

  `https://bioinfo2.ugr.es/TextKeywords/SpanishTexts/donquijote.txt`

cat i

Descargar El Quijote en texto plano

`https://bioinfo2.ugr.es/TextKeywords/SpanishTexts/donquijote.txt`

- ¿Qué hace `cat`?

1' `man cat`

cat i

Descargar El Quijote en texto plano

`https://bioinfo2.ugr.es/TextKeywords/SpanishTexts/donquijote.txt`

- ¿Qué hace `cat`?

1' `man cat`

Ejecutar y *diseccionar*

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

¿Qué ocurre?

cat i

Descargar El Quijote en texto plano

`https://bioinfo2.ugr.es/TextKeywords/SpanishTexts/donquijote.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
```

cat ii

- 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
```

Conectado 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"