

Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informàtica de Sistemes y Computadoras (DISCA)

Universitat Politècnica de València

Bloque Temático 1:
Concepto de sistema operativo

Seminario 2 Introducción a UNIX

fSO

DISCA



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

- **Objetivos:**

- Explicar la **utilidad y funcionamiento del shell** de UNIX
- Describir la **estructura básica del sistema de archivos en UNIX**
- Exponer el **concepto de ruta (*path*)** de acceso a archivos
- Describir **las variables y órdenes del shell** más utilizadas

- **Bibliografía clásica**

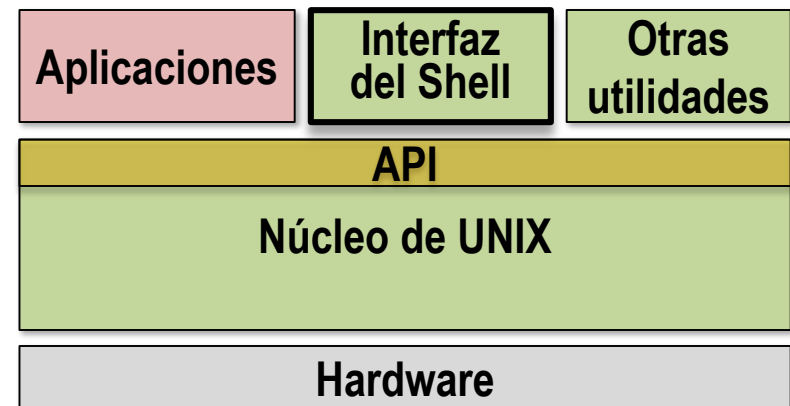
- Kernighan, Pike: The UNIX programming environment/El entorno de programación UNIX, Prentice Hall (poliformat: Material Complementario / Docs)

- **Accesible en Internet**

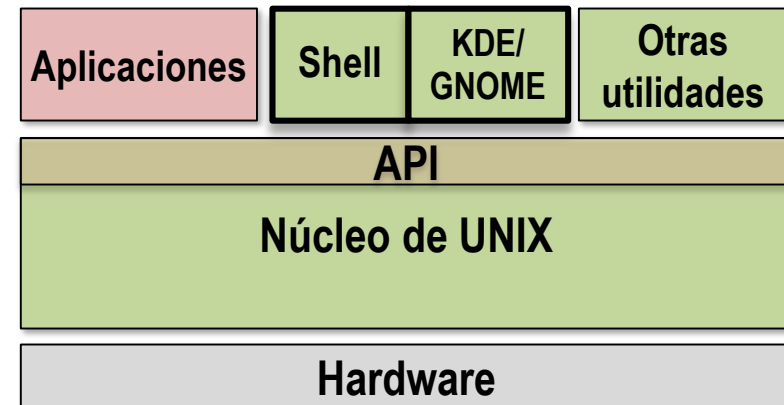
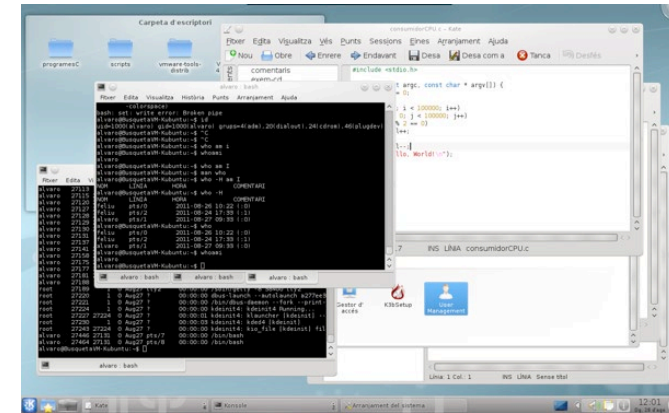
- Mark Burgess: The UNIX programming environment
http://www.iu.hio.no/~mark/unix/unix_toc.html
- Fernando Bellas: El entorno de programación UNIX
<http://www.tic.udc.es/~fbellas/teaching/unix/index.html> y (poliformat: Material Complementario / Docs)

- **Introducción**
- Identidades en UNIX
- Variables del Shell
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- Manejo de la entrada/salida
- Manejo de procesos
- Programación con el Shell

- **Interfaces de usuario (interfaces de texto o gráficas):**
 - Necesarias para ejecutar programas, organizar archivos, administrar el sistema, etc.
 - **Programa** que traduce las acciones del usuario en llamadas al núcleo de UNIX, a través de la interfaz a los programas (API), y muestra los resultados de ejecución
 - **Interfaz de texto clásica:** consola formadas por un teclado y un monitor en modo texto
 - Dentro de UNIX: dispositivos *tty0*, *tty1*,...
 - Lenguaje: el **shell** de UNIX



- **Interfaces gráficas de UNIX o GUI**
 - *X Window* (1984): *X11* o simplemente *X*
 - Implementaciones:
 - *XFree86*, *X.Org* (≥ 2004)
 - *X Quartz*
 - *Cygwin/X*, *mobaXterm*, *Xming*, ...
 - *FreeNX*
 - *Android X Server*
 - Está pensada para monitor en modo gráfico, teclado y ratón
 - **Emulan una o más consolas** de texto dentro de una ventana como pseudoterminales (dispositivos pts/0, pts/1, ...)
 - Distribuida: es posible conectarse desde máquinas diferentes a la que la inicia.
 - KDE, GNOME y otras **ofrecen las herramientas de uso y administración** mediante la GUI



Nomenclatura:

KDE: powerful graphical desktop environment for UNIX workstations

GNOME: GNU Object Model Environment

- **Shell de UNIX**

- **Intérprete de órdenes: programa que lee líneas de órdenes, las interpreta y las ejecuta**
- La sintaxis del *shell* permite especificar secuencias de órdenes a la manera de un lenguaje de programación
 - tiene comentarios, variables, control de flujo, funciones (con argumentos y valores de retorno), etc.
 - **Shell script:** archivos de texto que contienen líneas de órdenes
- Hay diversos *shells* (parecidos pero no iguales): *cshell*, *kshell*...
 - vamos a estudiar el **bash o Bourne Again Shell**
- El shell de UNIX permite:
 - gestionar el sistema de archivos
 - ejecutar programas
 - direccionar la entrada/salida de los programas
 - mantener variables del sistema
 - monitorizar y gestionar los procesos

- **Diálogo del shell**

repetir

escribir el *prompt*

leer la línea escrita por el usuario

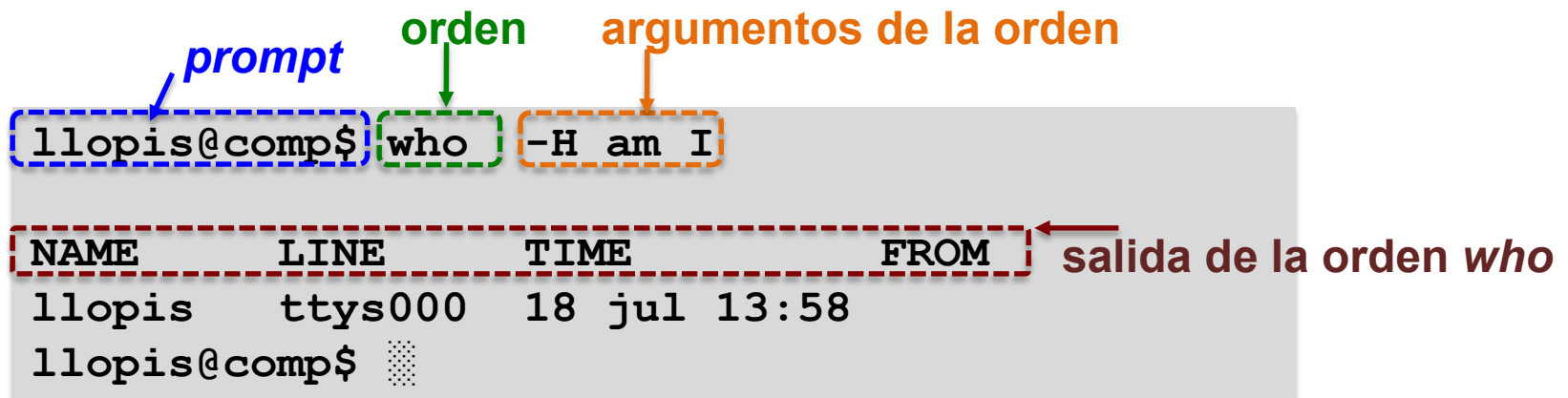
procesar la línea (desarrollando los metacaracteres)

identificar la orden (el primer *token* de la línea)

ejecutar la orden transfiriéndole argumentos (resto de *tokens*)

recibir el código de terminación del programa

hasta (final del archivo de entrada)



Metacaracteres: `$ * ? \ > < | & []`

- **Órdenes internas y externas**
 - El *shell* trata dos tipos de órdenes:
 - Órdenes internas o *built-in commands*:
 - El *shell* incluye internamente el código que hace el trabajo
 - Órdenes externas: el *shell* busca un programa en el disco y lo ejecuta
 - Archivos con código binario (ejecutable), p. ej. C y compilados
 - *Shell scripts* y otros *scripts* (*PHP*, *Python*, *Perl*, ...)
- **Ayuda en línea (*on line*): orden *man***
 - Descripción de una orden concreta: `man orden`
 - Descripción del *shell*: `man bash`

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

NAME
    ls -- list directory contents

SYNOPSIS
    ls [-ABCFGHLPRTWZabcdefghiklmnopqrstuwX1] [file ...]

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


- Introducción
- **Identidades en UNIX**
- Variables del Shell
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- Manejo de la entrada/salida
- Manejo de procesos
- Programación con el Shell

- **Usuarios y grupos:** son la base del esquema de protección de UNIX
 - Su definición es una tarea administrativa muy importante en UNIX
 - Un **usuario** es una identidad conocida por el sistema que tiene contraseña (*password*), privilegios, directorio inicial (*home*), etc.
 - cada usuario tiene un nombre (USERNAME) y un número (**UID**)
 - hay usuarios de sistema: *daemon*, *mail*, etc.
 - *root*: es un usuario predefinido que tiene los máximos privilegios
 - Un **grupo** es un conjunto de usuarios
 - cada grupo tiene un nombre y un número (**GID**)
 - hay grupos predefinidos: *adm*, *man*, etc.
 - cada usuario está necesariamente en un grupo primario

Nomenclatura:**UID:** User Identification**GID :** Group Identification

- Órdenes

Orden	Descripción de utilidad
id	muestra la identidad y los grupos a los que pertenece el usuario
su	cambia de usuario
who	lista usuarios conectados actualmente

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

- Introducción
- Identidades en UNIX
- **Variables del Shell**
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- Manejo de la entrada/salida
- Manejo de procesos
- Programación con el Shell

- Qué son las variables del *shell*?
 - Son símbolos que tienen asignado una cadena de caracteres
 - En la línea de órdenes: un símbolo de la forma *\$nombre_de_variable* se substituye por el valor de la variable
 - = define y asigna valor a una variable en la línea de órdenes *nombre_de_variable=valor* (importante: sin espacios)

```
...$ MiNombre=Feliu
...$ echo MiNombre
MiNombre
...$ echo $MiNombre
Feliu
```

- El entorno (*environment*)
 - El conjunto de variables accesibles desde programas externos se llama **entorno**.
 - Una variable de entorno se define con la orden **export**
 - **env**: orden que lista el valor de todas las variables del entorno

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

- Órdenes para visualizar variables del *shell* y definir variables

Orden	Descripción de utilidad
=	Crea una variable y le asigna valor (nombre=valor)
export	Marca una variable como variable de entorno
env	Lista el valor de todas las variables de entorno
echo	Muestra una cadena de caracteres por la salida estándar

- Algunas variables del *shell*

Variables Shell	Descripción de utilidad	Orden mostrar valor
PATH	Contiene la ruta de búsqueda de programas	echo \$PATH
HOME	Contiene ruta del directorio del usuario	echo \$HOME
HOSTNAME	Nombre del computador en la red	echo \$HOSTNAME
PS1	Aspecto del <i>prompt</i>	echo \$PS1
PWD	Directorio actual de trabajo	echo \$PWD
SHELL	Intérprete de órdenes por defecto	echo \$SHELL
TERM	Tipo de terminal	echo \$TERM

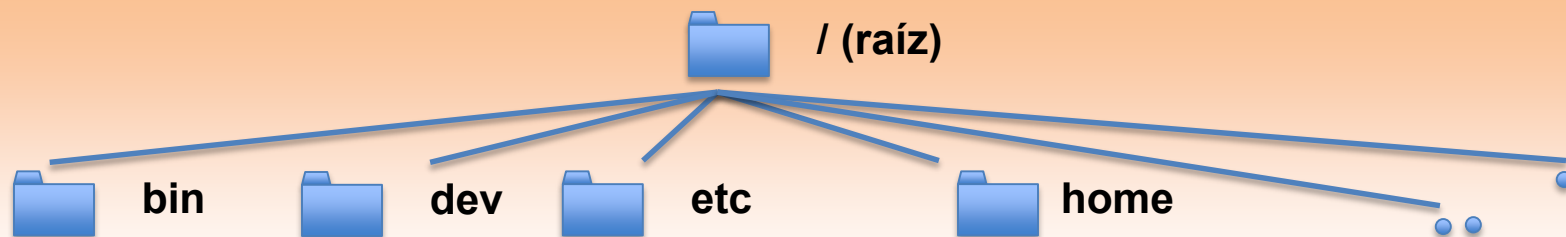
- La variable ***PATH***

- Contiene la lista de directorios donde el *shell* busca los programas y scripts al procesar la línea de órdenes:
 - los nombres de directorio están separados por “.”
 - al buscar los programas, el intérprete hace el recorrido por orden

```
...$ echo $PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
...$ export PATH=$HOME/pruebas:$PATH
...$ echo $PATH
/home/llopis/pruebas:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
...$
```

- Introducción
- Identidades en UNIX
- Variables del Shell
- **Navegación por el sistema de archivos**
- Manejo del sistema de archivos
- Manejo de la entrada/salida
- Manejo de procesos
- Programación con el Shell

- **Esquema del sistema de archivos UNIX**

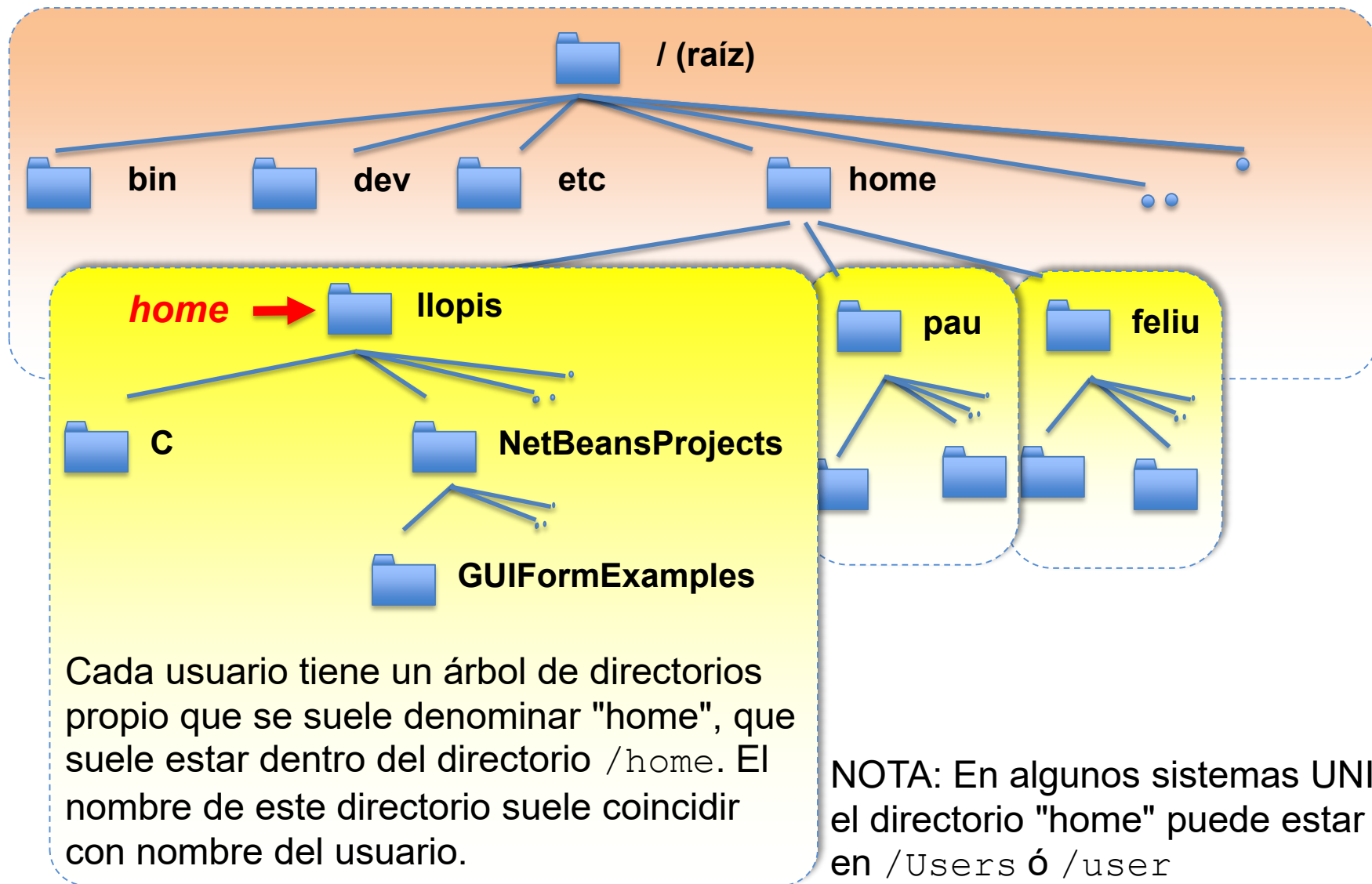


- Directorios de uso general
 - Contienen los programas y los datos útiles para el sistema y para todos los usuarios
- Directorio “.”
 - Hace referencia al directorio actual de trabajo
- Directorio “..”
 - Hacer referencia al directorio “padre”

algunos ejemplos

directorio	Tipo de archivos que contiene
/bin	órdenes comunes
/dev	dispositivos de entrada/salida
/etc	información de administración
/usr	aplicaciones

- Esquema de directorios en UNIX



- **Nombres de archivo y directorio en UNIX**

- Los directorios forman **un árbol** (invertido) que crece desde la raíz
 - El nombre completo (camino o *path*) de un archivo o directorio contiene el nombre de todos los directorios superiores
 - El separador es el carácter “/”
 - El nombre del **directorio raíz** es “/”
 - **Nombre absoluto** comienza desde la raíz del sistema de archivos
 - Ejemplo: “/home/llopis/eclipse/p1/main.java”
 - **Nombre relativo** hace referencia desde el directorio actual de trabajo
- Casos especiales:

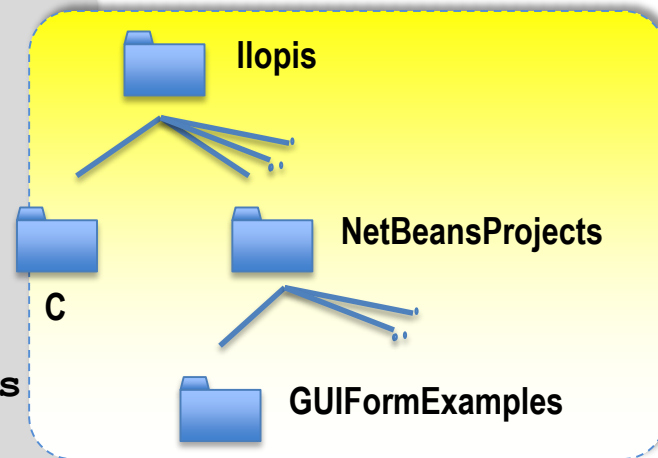
- “.” es el directorio de trabajo actual
- “..” es el directorio **padre** del directorio actual de trabajo
- Ejemplo: situados en el directorio “/home/llopis/eclipse”
 - “..” es el directorio “/home/llopis”
 - “.” es el directorio “/home/llopis/eclipse”

- **Directorio de trabajo**

Orden	Argumentos	Utilidad
<code>cd</code>	<i>directorio</i>	Fija el directorio de trabajo
<code>pwd</code>		Muestra el directorio de trabajo actual

- `cd [dir]`: cambia el directorio de trabajo al directorio *dir*
 - sin argumentos: cambia el directorio de trabajo al directorio *home*
- `pwd`: muestra el camino absoluto del directorio de trabajo
- al inicio de una sesión, el directorio de trabajo es *HOME*

```
...$ pwd
/home/llopis
...$ cd NetBeansProjects
...$ pwd
/home/llopis/NetBeansProjects
...$ cd GUIFormExamples
...$ pwd
/home/llopis/NetBeansProjects/GUIFormExamples
...$ cd
...$ pwd
/home/llopis
...$
```



- Introducción
- Identidades en UNIX
- Variables del Shell
- Navegación por el sistema de archivos
- **Manejo del sistema de archivos**
- Manejo de la entrada/salida
- Manejo de los procesos
- Programación con el Shell

- Órdenes del *shell* para el manejo del sistema de archivos

Orden	Opciones	Argumentos	Utilidad
cat		<i>archivo(s)</i>	vuelca el contenido de los archivos
ls	-l -a -d	<i>archivo(s)</i>	lista información variada sobre el archivo
file		<i>archivo(s)</i>	informa del tipo de archivo
rm	-R	<i>archivo(s)</i>	elimina enlaces o archivos
mkdir		<i>directorio</i>	crea un directorio
rmdir		<i>directorio</i>	elimina un directorio
mv		<i>archivo archivo archivo(s) dir</i>	renombra/reubica archivos
cp	-r	<i>archivo archivo archivo(s) dir</i>	copia archivo(s)
ln	-s	<i>archivo archivo archivo dir</i>	crea nuevos enlaces a un archivo existente
chown	-R	<i>usuario archivo(s)</i>	fija el propietario
chmod	-R	<i>modo archivo(s)</i>	fija permisos

– more, less, locate, which, find, df, du, ...

- **Orden cat:** Muestra por pantalla el contenido de un archivo
Uso: *cat [nombre archivo]*
 - No aplicable a directorios
- **Orden ls:** Lista nombres y otros atributos de los archivos:
Uso: *ls [opciones] [nombres...]*
 - *opciones:* detalles que da de cada archivo
 - **l** (de *long*): permisos, propietario, grupo, ...
 - **d** muestra información de los directorios
 - **a** muestra todos los nombres de archivo incluidos los ocultos
 - *nombres:* puede ser de directorios o de archivos
 - si *nombres* es de directorio: se listan los archivos que contiene (a no ser que se use la opción **-d**)
 - si no se especifica un nombre: se lista el directorio de trabajo
- Tratamiento especial de **nombres ocultos**
 - Los archivos cuyos nombres comiencen por punto (.) están ocultos y no aparecen en el listado (si aparecen con opción **-a**)

- **Patrones de nombres de ficheros → comodines**
 - Son caracteres que sustituyen cadenas de caracteres
 - Sirven para nombrar a varios ficheros sin explicitarlo
 - Cuando un *token* contiene un comodín, el *shell* hace *pattern matching* con las entradas de un directorio y sustituye el *token* por todos los nombres que se ajustan al patrón
 - Ejemplos
 - ‘*’ se corresponde con cualquier secuencia de caracteres
 - ‘?’ se corresponde con cualquier carácter, uno sólo carácter

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


- Formato largo de la orden `ls` (opción `-l`)

bloques de disco ocupados

```
...$ ls -al
```

total	56						
drwxr-xr-x	7	llopi	fso	238	28 jun 20:51	.	
drwxr-xr-x	9	llopi	fso	306	26 jun 16:50	..	
-rwxr-xr-x	1	llopi	fso	12612	26 jun 08:02	a.out	
drwxr-xr-x	3	llopi	fso	102	28 jun 20:51	datos	
-rw-r--r--	1	llopi	fso	316	26 jun 08:02	prog.c	
-rw-r--r--	1	llopi	fso	66	28 jun 20:23	error.txt	
-rw-r--r--	1	llopi	fso	12	28 jun 20:23	lista.txt	

```
..$
```

permisos

tipo de archivo

archivo regular
directorío
(otros)

número de enlaces

propietario

grupo

tamaño (en bytes)

fecha de modificación

nombre

- **Archivos: propietarios y permisos**

- Un archivo es propiedad de un usuario (UID) y de un grupo (GID)
 - Inicialmente, los propietarios son el usuario que lo ha creado y el grupo primario al que pertenece el usuario
 - La orden **chown** (*change owner*) permite cambiar los propietarios
- Tres **permisos** independientes: **r** (*read*), **w** (*write*), **x** (*execute*)

permiso	archivos regulares	directorios
r	el archivo se puede leer	el directorio se puede listar con ls (sólo nombres)
w	el archivo se puede escribir	se pueden eliminar o añadir archivos (necesita x)
x	el archivo se puede ejecutar (si es binario o un script)	se puede usar (acceso a los archivos, usarlo como directorio de trabajo). Para ls -l es necesario tener además permiso r.

- Tres ámbitos de **propiedad**: **user**, **group** y **other**
- Aplicación de los permisos en el acceso del usuario *U* al archivo o directorio *X*:
 - si** (*U* es el root) permiso concedido
 - sino si** (*U* es el propietario de *X*) aplica permisos *user*
 - sino si** (*U* está en el grupo propietario de *X*) aplica permisos *group*
 - sino** aplica permisos *other*

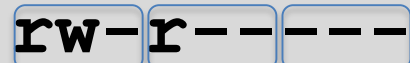
- Gestión de permisos con **chmod** (*change mode*)
 - 9 bits : 3 ámbitos (*user, group, other*) x 3 permisos



rwxrwxrwx

user group other

- Modo octal o numérico:
 - Tres dígitos. Cada dígito codifica los tres permisos de un ámbito con valores del 0 al 7
 - Ejemplo: `chmod 640 nombre`



rw-r---

user group other

- Modo simbólico
 - Modifica permisos uno a uno
 - Ejemplos:
 - `chmod u+w nombre`
 - `chmod +x nombre`
 - `chmod a+r,o-w nombre`
 - `chmod ug=rw,o= nombre`

a quién	operación	permiso
u (user)	+ (añadir)	r
g (group)	– (quitar)	w
o (other)	= (fijar)	x
a (all)		

- Introducción
- Identidades en UNIX
- Variables del Shell
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- **Manejo de la entrada/salida**
- Manejo de los procesos
- Programación con el Shell

- **Dispositivos estándar**

- El *shell* maneja **tres canales** de caracteres:
 - Entrada estándar ***stdin*** (canal 0)
 - Salida estándar ***stdout*** (canal 1)
 - Salida de errores estándar ***stderr*** (canal 2)
- Cada canal esta asignado a un dispositivo: la consola, un archivo u otros
- Por omisión, en una sesión interactiva, los tres canales están asignados a la consola
- El *shell* escribe el *prompt* en *stdout* y lee la línea de órdenes de *stdin*
- Los programas pueden leer de *stdin* y escribir en *stdout* y *stderr*
- Ejemplo (ls):

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

← salida estándar

← salida de errores

- **Redireccionamiento**

- Redireccionar: consiste en asignar la entrada (*stdin*) y/o las salidas estándar (*stdout*, *stderr*) a un dispositivo concreto (típicamente a un archivo)
- Posibilidades desde la línea de órdenes:

forma	Tipo de redirección que se realiza
< dispositivo	redirecciona <i>stdin</i> al dispositivo
> dispositivo	redirecciona <i>stdout</i> al dispositivo (sobrescribe si existe)
>> dispositivo	redirecciona <i>stdout</i> al dispositivo (añadiendo al final si ya existe)
2> dispositivo	redirecciona <i>stderr</i> al dispositivo (creándolo nuevo si ya existe)
2>&1	redirecciona <i>stderr</i> al dispositivo asociado a <i>stdout</i>

```
...$ ls p a*
ls: p: No such file or directory
a.out
...$ ls q a* > lista.txt 2> errores.txt
...$ cat lista.txt
a.out
...$ cat errores.txt
ls: q: No such file or directory
...$
```

stdout redireccionada a *lista.txt*

stderr redireccionada a *errores.txt*

- **Filtros**

- Órdenes que leen de *stdin*, hacen operaciones sencillas y escriben el resultado en *stdout*

Orden	Opciones	Argumentos	Descripción de utilidad
head	-n lin		transcribe las <i>n</i> primeras líneas leídas
tail	-n lin		transcribe las <i>n últimas</i> líneas leídas
sort			ordena las líneas de texto leídas y escribe el resultado
tee		<i>archivo</i>	transcribe la entrada en la salida y en un archivo
wc	-l -w -c		cuenta líneas, palabras y caracteres leídos y escribe la estadística
grep		<i>regex</i>	transcribe las líneas que satisfacen una expresión regular
awk			procesa archivos buscando patrones
cut	-f -d	<i>regex</i>	selecciona componentes de cada línea que procesa
sed		<i>script archivo</i>	editor de flujo de caracteres

- “*regex*” (expresiones regulares) :describen un conjunto de cadenas que contienen un patrón. Usa metacaracteres como `\ ^ $. [] { } | () * + ?`
- Pueden encadenarse: en secuencia con ‘;’ o en conexión con ‘|’

- Filtros (ejemplos)

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

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

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

```
...$ head -n 3 < entrada; grep fi < entrada
...$ grep fi <entrada; echo "Hay`wc -l entrada` coincidencia/s"
```


- **Tuberías (*pipe*)**

- Permite la comunicación entre procesos
- Conectan la salida estándar de una orden a la entrada estándar de la orden siguiente

```
...$ sort <entrada | head -n 3
five
four
one
...$ sort <entrada | tail -n 3 >salida
...$ cat salida
one
three
two
```

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

```
...$ df | sort -rnH|head -1
...$ ps -ax | grep firefox | cut -f 1 -d " "
```

- Introducción
- Identidades en UNIX
- Variables del Shell
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- Manejo de la entrada/salida
- **Manejo de procesos**
- Programación con el Shell

- **Procesos en UNIX**

- Se identifican por su **PID** (*process identifier*)
 - el *shell* muestra su propio PID con `echo $$`
- Cada proceso está asociado a un usuario con UID dado
- El conjunto de procesos tiene estructura de árbol
 - Cada uno tiene un **proceso padre** definido por su **PPID** (*parent process identifier*)
 - Cada uno puede crear un **proceso hijo** (*child process*) o más

- Órdenes:

orden	opciones	argumentos	utilidad
ps	-e -f a f	<i>pid(s)</i>	lista información de los procesos
kill	-s -n	<i>señal pid(s)</i>	envia una señal a los procesos
sleep		<i>tiempo</i>	suspende la ejecución del intérprete
pstree			muestra el árbol de procesos
top htop			muestran estadísticas de los procesos en tiempo real

- Orden `ps -ef`

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

usuario

identidad del proceso

uso de CPU (%)

fecha/hora de creación

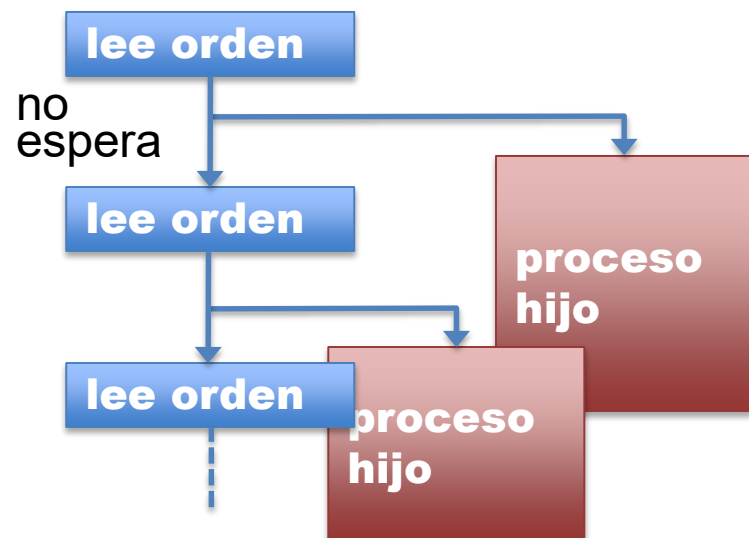
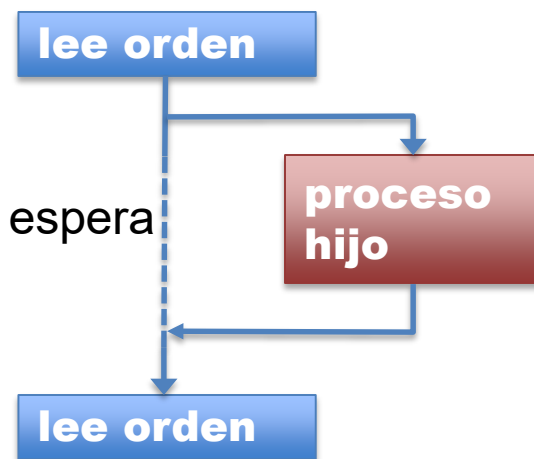
entrada de control

tiempo de CPU acumulado

orden o programa

- Muestra la relación padre→hijo entre los procesos
 - arranque (PID 0) → init (PID 1), kthread (PID 2)
 - init (PID 1) → kdeinit (PID 19892) → bash (PID 19894) → ps -ef (PID 19914)

- **Procesos en primer plano (interactivo o *foreground process*) y en segundo plano (*background process*)**
 - El SO crea un nuevo proceso (proceso hijo del *shell*) para ejecutar una orden externa
 - Procesos en **primer plano** (por omisión): el *shell* espera que finalice el proceso hijo antes de continuar su ejecución y mostrar *prompt*
 - El proceso hijo se puede interrumpir con ctrl-C (^C)
 - Proceso en **segundo plano**: el *shell* y el proceso hijo continúan su ejecución concurrentemente
 - *orden* & : orden se ejecuta en *background*
 - *kill -9 PID* : El proceso con identificación PID finaliza



- Ejemplos

```
...$ sleep 100 &
[1] 41187
...$ ps
  PID TTY          TIME CMD
 2050 ttys001    0:00.69 -bash
 41187 ttys001    0:00.00 sleep 100
...$ kill -9 41187
...$ ps
  PID TTY          TIME CMD
 2050 ttys001    0:00.70 -bash
[1]+  Killed                  sleep 100
...$
```

Proceso en segundo plano

Dos procesos: el shell y el programa *sleep*

Orden de terminación del proceso *sleep*

Sólo permanece el shell

Mensaje de final de ejecución de *sleep*

- Introducción
- Identidades en UNIX
- Las variables del Shell
- Archivos y directorios
- Navegación por el sistema de archivos
- Manejo del sistema de archivos
- Control de la entrada/salida
- Manejo de procesos
- **Programación con el Shell**

- *Shell scripts*
 - Son archivos de texto, formados por órdenes del *shell*
 - Aceptan argumentos
 - Se ejecutan:
 - mediante la orden sh: **sh nombre argumento(s)**
 - invocándolos directamente por su nombre
 - si no están ubicados en ninguno de los directorios listados en PATH, habrá que dar su ruta (absoluta o relativa a PWD)
por ejemplo `$./nombre`
 - han de tener permiso de ejecución -x
 - Heredan parte del entorno del *shell*
 - Los argumentos del script son accesibles como variables:

Símbolo de variable	Argumento del script
\$0	nombre del script
\$1...\$9	1º ... 9º argumento
\$#	número de argumentos

```
#!/bin/sh
if[$# -gt 0 ]
then
    echo "param1 es $1"
else
    echo "Uso $0 param1"
fi
```


- Código de terminación (*exit code*) de las órdenes
 - La variable denominada `?` contiene el código de terminación de la última orden
 - este valor numérico se muestra con **`echo $?`**
 - Código de retorno = 0 : ejecución de la orden sin errores
 - Se interpreta como, 0 = cierto y cualquier otro valor como falso
 - Código de retorno = un número entre 1 y 255 dependiendo del fallo de la orden
- Orden **`exit`**: finaliza la ejecución del script
- Órdenes relacionadas con el código de terminación

```
...$ ls
lote          lote-llarg...
...$ echo $?
0
...$ ls otro-nombre
ls: otro-nombre: No such fi...
...$ echo $?
1
...$
```

orden	Descripción de utilidad
true	no hace nada y devuelve un código de terminación 0
false	no hace nada y devuelve un código de terminación 1
test	evalúa una condición: devuelve código de terminación 0 si se cumple y 1 en caso contrario

- Ejemplos

```
## creadir: script con un argumento
## Crea un directorio vacío de nombre dado por el argumento $1

# Si no existe ningún archivo ni directorio con el nombre, lo crea
if ! test -e $1; then mkdir $1; exit 0;
# Si existe el directorio, borra su contenido
elif test -d $1; then rm -r $1/*; exit 0;
# En otro caso, no hace nada y señala error
else echo $1 ya existe y no es un directorio; exit 1; fi
```

```
## allold: script sin argumentos
## Añade la terminación ".old" a todos los nombres de archivo
## del directorio actual

# i es una variable local del script que valdrá cada nombre
# del directorio actual
for i in * ; do mv $i $i.old; done
exit 0
```

```
$ for((i=1000;i--; i>0)); do echo "$i horchatas. Solo quedan $i
horchatas"; done
```

- **Ejercicio SUT2.1_** Ejecute una a una las siguientes líneas de órdenes del *shell* en una máquina UNIX, analice el resultado de la ejecución de cada una y conteste a las preguntas propuestas:

```
$ cat prueba  
$ echo "Hola soy un alumno de FSO" >&1  
$ echo "Hola soy un alumno de FSO" >prueba  
$ cat prueba  
$ echo "Estoy practicando el shell" >>prueba  
$ cat prueba | wc -l
```

Conteste para cada línea de órdenes:

- a) ¿Cuál es el resultado de ejecución?
- b) ¿Se ha creado un nuevo archivo?
- c) ¿Se ha ejecutado correctamente la línea de órdenes?
- d) ¿Que código de retorno ha devuelto el sistema?
- e) ¿Cuántas órdenes intervienen en esta línea?
- f) ¿Cuántos archivos intervienen en esta línea?

¡Aviso!: el `$` que encabeza cada orden representa el *prompt* de la máquina UNIX y no hay que escribirlo.