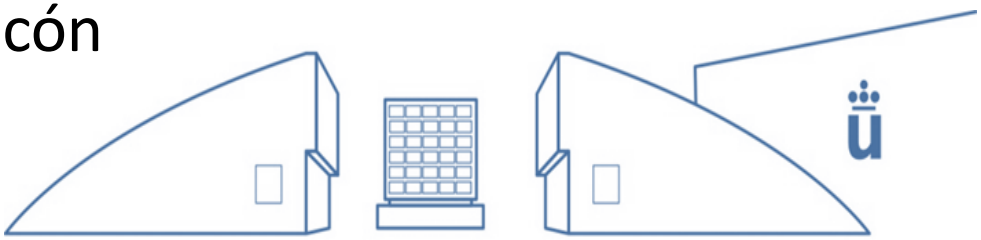
Tema 1

Intérprete de mandatos

Alberto Sánchez

Sofia Bayona

Luis Rincón



Bibliografía

- **“Sistemas Abiertos”** *Luis José Cearra. ETSISI-UPM, 2014.*
  - Todo el libro excepto los capítulos 8, 9, 17, 18, 19, 22, 23 y 24 y apéndice.
  - Los capítulos no entran completos.
  - El libro se puede usar como referencia para consultar los mandatos estudiados a lo largo del tema.

Índice

Intérprete de mandatos

- Un intérprete de mandatos o *shell* es un programa que hace de interfaz de texto entre el usuario y el Sistema Operativo.
- La *shell* tiene la función de interpretar mandatos y lanzar programas.
- La *shell* también proporciona funcionalidades de programación en forma de *scripts*.
- Intérpretes de mandatos más habituales:
  - *cmd.exe, powershell.exe* en sistemas Microsoft Windows.
  - *sh, csh, tcsh, ksh, bash . . .* en sistemas tipo UNIX.

Intérprete de mandatos

- Los entornos de escritorio proporcionar acceso a los programas de *shell*: *terminal, konsole, xterm*.

Intérprete de mandatos

- La *shell* es una aplicación que forma parte del S.O. y ofrece una interfaz de la cual usuarios y administradores pueden operar con el sistema.
  - Consultar el estado del sistema.
  - Recorrer el árbol de directorios.
  - Lanzar otros mandatos o aplicaciones.
    - Los mandatos y aplicaciones se invocan usando el nombre del fichero ejecutable.
    - El ejecutable tiene que estar dentro del PATH.
    - Si no está dentro del PATH, hay que dar su ruta de acceso.
- Sintaxis típica:

```
(prompt) mandato [parámetro 1] ... [parámetro n]
```

- Ejemplo:

```
$ echo "Hola Mundo"
```

Intérprete de mandatos

- Funcionamiento:
  1. Muestra el *prompt* (habitualmente es el símbolo $, que puede ir precedido del nombre de usuario, directorio actual, etc).
  2. Lee una línea.
  3. La analiza.
  4. Evalúa los mandatos de la línea.
  5. Espera a que terminen (o no).
  6. Vuelve a 1.
- Se pueden introducir varios mandatos en una misma línea, separados por “;”

```
$ echo "Hola"; echo ; echo "Mundo”
Hola

Mundo
```

Ayuda: mandato man

- Mandato **`man`** (texto) / **`xman`** (gráfico)
- El manual de UNIX contiene información sobre la mayoría de mandatos, ficheros y servicios del sistema
  - **`man`** ***`mandato`***
- Es solo un visor de texto, concretamente **`less`**. Para salir del manual: "**`q`**"
- Las páginas del manual se organizan por secciones:
  1. Mandatos generales
  2. Llamadas al sistema
  3. Biblioteca de C
  4. Ficheros especiales
  5. Formatos de ficheros
  6. Juegos y salvapantallas
  7. Otros
  8. Administración

less

Keystroke	Action
h or H	Displays help on using less.
Page Down, spacebar, Ctrl+V, f, or Ctrl+F	Moves down one screen in the document.
Page Up, Esc+V, b, or Ctrl+B	Moves up one screen in the document.
Down Arrow, Enter, Ctrl+N, e, Ctrl+E, j, or Ctrl+J	Moves down one line in the document.
Up Arrow, y, Ctrl+Y, Ctrl+P, k, or Ctrl+K	Moves up one line in the document
xg, x<, or x Esc+<	Go to line x in the document—for instance, typing 100g displays the document's 100th line. If x is omitted, the default is 1.
xG, x>, or x Esc+>	Go to line x in the document. If x is omitted, the default is the last line of the document.
xp or x%	Go to the point x percent through the document—for instance, typing 50p goes to the document's halfway point.
/pattern	Searches forward for pattern in the document, starting at the current location.
?pattern	Performs a backwards search, locating instances of pattern before the current location.
n or /	Repeat the previous search.
q or Q or :q or :Q or ZZ	Quits from the less pager.

man: páginas de manual

- Si un mandato aparece en varias secciones, se puede indicar qué sección queremos ver:

  man *seccion mandato*

- Ejemplo: el mandato **write** existe como mandato de usuario y como llamada al sistema:

  man write
  man 2 write

- Para buscar una página del manual:
  - **whatis <palabraclave>** : búsqueda por nombre.
  - **apropos <palabraclave>** : búsqueda exhaustiva. Igual que **man –k <palabraclave>**

info

- Algunas herramientas en vez de página **`man`** tienen página **`info`**. Apoyado por FSF.
- En lugar del formato lineal de las páginas **`man`**, las páginas **`info`** están organizadas en una estructura en árbol con menús y referencias cruzadas mediante hipertexto.
  - Organizado en niveles a partir de **nodos** (cada página **`info`** individual).
- Acceso a **`info`** de un mandato: **`info <mandato>`**

info: órdenes

Keystroke	Action
?	Displays help information.
N	Moves to the next node in a linked series of nodes on a single hierarchical level. This action may be required if the author intended several nodes to be read in a particular sequence.
P	Moves back in a series of nodes on a single hierarchical level. This can be handy if you've moved forward in such a series but find you need to review earlier material.
U	Moves up one level in the node hierarchy.
Arrow keys	Moves the cursor around the screen, enabling you to select node links or scroll the screen.
Page Up, Page Down	These keys scroll up and down within a single node, respectively. (The standard info browser also implements many of the more arcane commands used by less and outlined in Table 8.2.)
Enter	Moves to a new node once you've selected it. Links are indicated by asterisks (*) to the left of their names.
L	Displays the last info page you read. This action can move you up, down, or sideways in the info tree hierarchy.
T	Displays the top page for a topic. Typically this is the page you used to enter the system.
Q	Exits from the info page system.

Índice

bash (Bourne-Again Shell)

- Los mandatos se invocan usando el nombre del fichero ejecutable.
  - El ejecutable tiene que estar dentro del PATH. Si no, entonces hay que indicar la ruta de acceso al ejecutable.
- Si no se indica fichero, suelen operar con la entrada estándar.
  - Se pueden redirigir la entrada y salida estándar del proceso arrancado.
- Se puede invocar más de un mandato en secuencia.
- Se puede recuperar el valor de retorno devuelto por el mandato y operar con él.
  - El valor de retorno es un número entero.
  - La *shell* captura ese valor y lo guarda por si se requiere su uso posterior.
  - El valor de retorno 0 indica terminación satisfactoria.
  - Un valor distinto de 0 suele indicar que se ha producido algún error. Diferentes valores (1, 2...) se usan como códigos de error.
  - La mayoría de *shells* tratan además el valor de retorno 0 como valor lógico verdadero y cualquier otro como falso.

bash: funcionalidad adicional

Keystroke	Effect
Tab	Command Completion
Up arrow	Retrieves the previous entry from the command history.
Left arrow	Moves the cursor left one character.
Right arrow	Moves the cursor right one character.
Ctrl+A	Moves the cursor to the start of the line.
Ctrl+E	Moves the cursor to the end of the line.
Ctrl+R	Searches for a command. Type a few characters and the shell will locate the latest command to include those characters. You can search for the next-most-recent command to include those characters by pressing Ctrl+R again.

Control de la ejecución

- La *shell* espera a que finalice la ejecución de un mandato antes de leer el siguiente salvo:
  - Si se lanzó el mandato en segundo plano (*background*) mediante el carácter "**&**":

    `mandato &`
  - Si se aborta la ejecución del mandato: combinación de teclas **^C** (*control + C*).
    - Habitualmente finaliza el mandato
  - Si se suspende la ejecución del mandato: combinación de teclas **^Z** (*control + Z*).
    - Suspender la ejecución de un mandato no lo finaliza. Se mantiene a la espera dentro del sistema

Control de la ejecución

- Los estados en que puede estar un proceso en la *shell* son:
  - En primer plano (*foreground*):
    - La *shell* espera a que termine la ejecución antes de ejecutar otro mandato.
  - En segundo plano (*background*):
    - Mientras se ejecuta el mandato se sigue teniendo el control de la *shell* y se pueden lanzar otros mandatos.
  - Pausado (suspendido):
    - El proceso esta parado debido a "**^Z**".

Control de la ejecución: mandatos

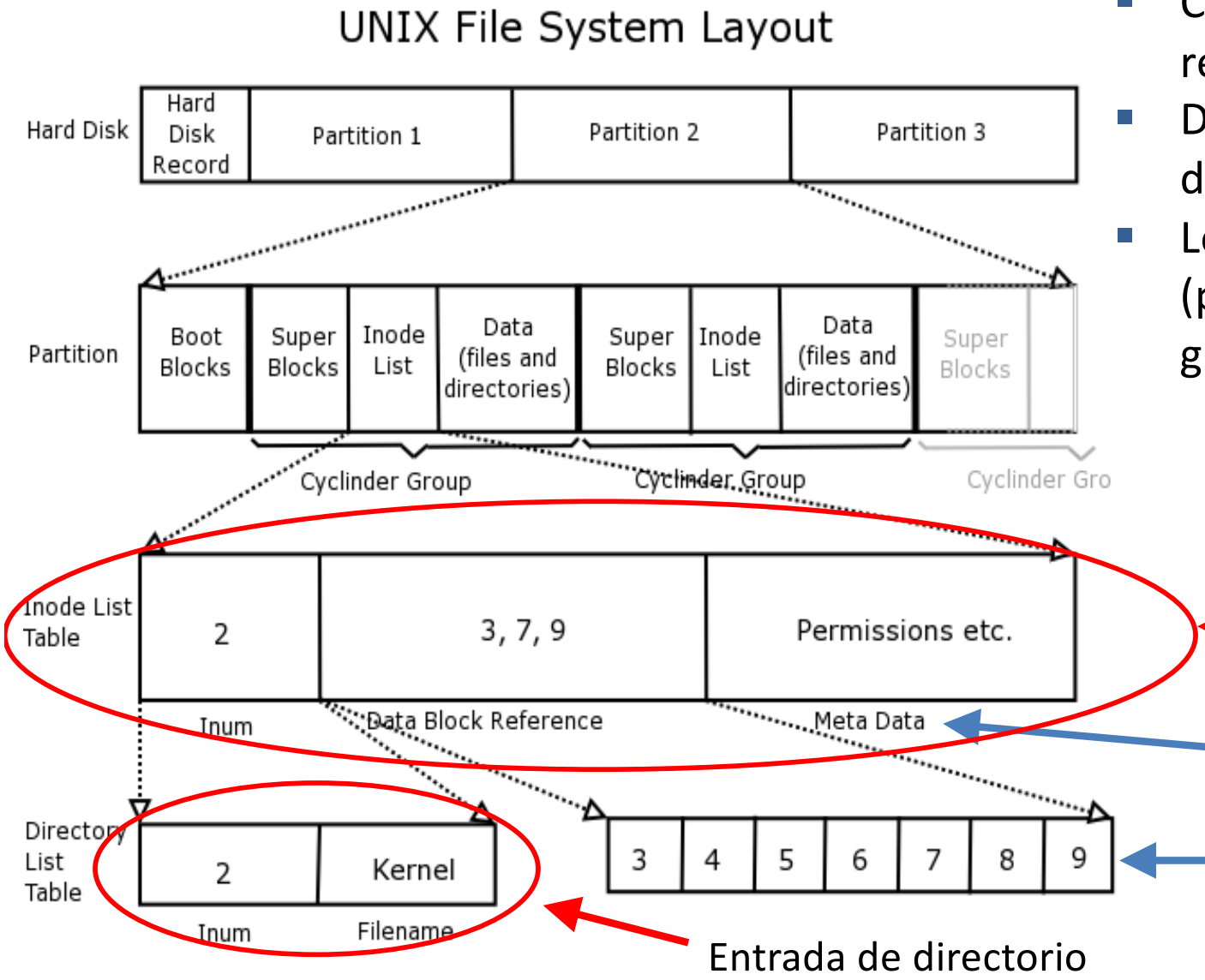
- `jobs` muestra una lista numerada de procesos en ejecución.
- **`bg`** reanuda la ejecución de un proceso pausado y lo pone en segundo plano.
- **`fg`** pasa un proceso pausado o en segundo plano a primer plano.
- **`ps`**: muestra la lista de procesos activos indicando su **pid**.
  - pid (*process identifier*): identificador del proceso (numérico).
  - **ps -a**: muestra todos los procesos asociados al terminal.
  - **ps -e**: muestra todos los procesos activos.
  - **ps -l**: muestra información en formato largo.
  - **ps aux**: muestra todos los procesos especificando usuario
- **`top`**: muestra dinámicamente la lista de procesos activos.
- **`kill`**: manda una señal a un proceso.
  - **kill -9 pid**: mandar señal al proceso pid para que aborte.

Mandatos internos y externos

- **Mandatos externos**: soportados por un fichero ejecutable.
  - Al invocar un mandato externo, la *shell* busca el ejecutable en los directorios del **PATH**, a no ser que se indique una ruta parcial o absoluta al fichero y lo ejecuta.
  - Al terminar la ejecución del mandato externo, la *subshell* muere y se vuelve a la original.
  - Ejemplos de mandatos externos: **ls**, **cp**, **rm**, **mkdir**, …
    - **whereis mandato** : localiza el ejecutable y la página de manual.
- **Mandatos internos** (*builtin commands*): implementados en la propia *shell*.
  - Al invocar un mandato interno, la *shell* ejecuta una función interna que lo implementa, sin crear una *subshell* y sin buscar ningún fichero ejecutable.
  - Ejemplos de mandatos internos: **cd**, **jobs**, **fg**, **bg**, …

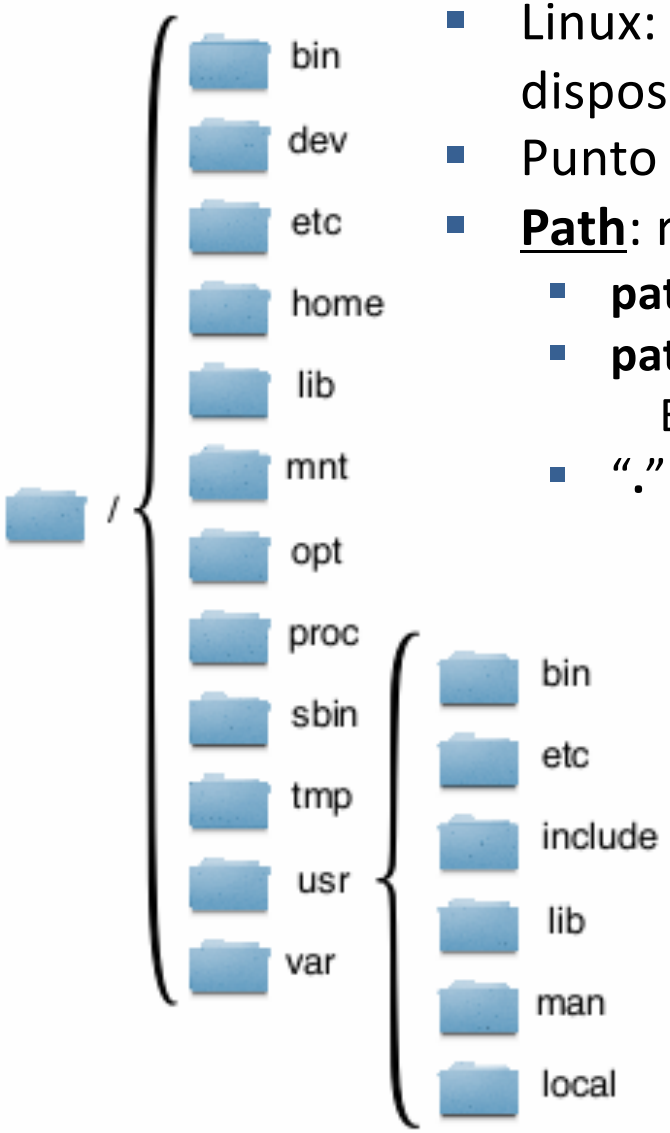
Índice

Conceptos básicos



- Cada fichero o directorio se representa por un i-nodo.
- Directorio: lista de ficheros o directorios.
- Los atributos de los ficheros (propietario, fecha, etc) se guardan en los i-nodos.

Árbol de directorios



- Linux: árbol jerárquico de directorios único con todos los dispositivos de almacenamiento de archivos.
- Punto inicial: directorio raíz /
- **Path**: ruta para encontrar un fichero
  - **path absoluto**: ruta completa. Ejemplo: **/usr/local/bin/bash**
  - **path relativo**: ruta a partir de un punto concreto.
    Ejemplo: **bin/bash** **../lib/modules**
  - "**.**" es el directorio actual y "**..**" el directorio padre.
    - Al iniciar sesión en la *shell* entramos en el directorio del usuario (**$HOME**): **/home/user**
    - **pwd**: imprimir directorio actual.
    - **cd**: cambiar de directorio (sin argumento cambia a directorio **$HOME**).
    - **`mkdir`**: crear directorio.
      - **–p**: crea directories intermedios que no están creados.
      - Ejemplo: **mkdir dir1/dir2**
    - **`rmdir`**: borrar directorio (tiene que estar vacío).
    - **`ls`**: listar directorio.
      - **-a**: muestra ficheros ocultos (su nombre empieza por "**.**")
      - **-l**: muestra detalles.

Montar un sistema de ficheros:

```
mount -t iso9660 /dev/hdc /media/cdrom
mount -t vfat /dev/hdb2 /media/dos
mount -t ntfs /dev/sda1 /media/windows
mount -t ntfs /dev/sda3 /home
```

Desmontar un sistema de ficheros:

```
umount /media/cdrom
```

Mostrar los sistemas de ficheros montados:

```
mount
```

Mostrar espacio libre/utilizado en sistemas de ficheros:

```
df
```

Mostrar espacio utilizado por los ficheros y directorios:

```
du
```

Mostrar árbol de directorios y ficheros:

```
tree
```

Ficheros – mandatos básicos

- **`touch`**: cambia fecha de modificación / crea un fichero vacío (**--date="fecha"**).
- **`cp`**: copia ficheros o directorios.
  - **-u**: solo actualiza los ficheros modificados.
  - **-r**: de forma recursiva.
- **`mv`**: renombra/mueve ficheros o directorios.
- **`rm`**: borra ficheros o directorios.
  - **-r**: borra directorios y su contenido.
  - **-f**: fuerza el borrado sin preguntar al usuario.
  - **-i**: pregunta antes de borrar. Útil para root.
- **`ln`**: crea un enlace de un fichero a otro.
  - *Hard link*: por defecto. Si el destino se mueve o se borra, el enlace sigue siendo válido.
  - *Symbolic link*: argumento **-s**. Si el destino se mueve o se borra, el enlace queda inválido (*broken link).*
- **`mktemp`**: crea fichero o directorio temporal.
- **`tar`**: une varios ficheros en un único archivo / restaura ficheros desde archivo.
  - Incluye opciones de compresión a través de gunzip, compress, etc.
  - Archivar: **tar cvzf archivo.tgz fichero1 fichero2 fichero3 …**
  - Restaurar: **tar xvzf archivo.tgz**

Ficheros – mandatos y filtros básicos

- **wc**: cuenta líneas, palabras y bytes de un fichero.
- **cut**: selecciona columnas específicas en ficheros de texto.
  - **-d 'símbolo'**: carácter utilizado para separar campos (separador).
  - **-fN** : N corresponde al número de la columna con la que quedarse.
- **head / tail**: selecciona las primeras / últimas líneas de un fichero.
- **sort**: ordena líneas en un fichero por orden alfabético.
  - **-t** simbolo **–KN:** para ordenar por una columna
  - **-r:** al contrario
  - **-u:** elimiina duplicados
- **tr**: sustituir o borrar caracteres.
  - **tr ":" "\t"**
- **sed**: sustituir.
  - sed s/original/new/ fichero. Acabado en /g sustituye todos los de la línea
  - sed /palabra/d fichero: Borra palabra
  - sed –Ii … fichero: Cambio persistente en el fichero
- **cat**, **less**, **more**: mostrar contenido de un fichero.
- **tee**: copia la entrada estándar en la salida estándar y en un fichero.
- **paste:** pega varios ficheros línea por línea.
- **od:** muestra el contenido de ficheros binarios
- **split**: parte un fichero en trozos con el mismo número de líneas.
- **cmp**, **diff**: comparan ficheros.
- **pico, nano**, **gedit**: editores de ficheros (NO SON FILTROS).

Ficheros – búsqueda: mandato grep

- **grep**: filtro que muestra la líneas de un fichero que presentan un patrón.
  - Trabaja sobre contenido de fichero. Si no se indica fichero opera con la entrada estándar.
  - El patrón puede ser expresado con una expresión regular

    -BN: muestra las N lineas anteriores la encontrada
  - Ejemplos: `grep root /etc/passwd`

    `grep '\<c..h\>' /usr/share/dict/words`

Option (long form)	Option (short form)	Description
--count	-c	Instead of displaying the lines that contain matches to the regular expression, display the number of lines that match.
--file=file	-f file	This option takes pattern input from the specified file rather than from the command line. The **fgrep** command is a shortcut for this option.
--ignore-case	-i	You can perform a case-insensitive search, rather than the default case-sensitive search, by using the -i or --ignore-case option.
--recursive	-R or -r	This option searches in the specified directory and all subdirectories rather than simply the specified directory. You can use **rgrep** rather than specify this option.
--extended-regexp	-E	Alternatively, you can call **egrep** rather than grep; this variant command uses extended regular expressions by default.

Expresiones regulares en grep

- Expresiones regulares básicas:
  - Expresiones con corchete: **b[ae]g, b[^ae]g**
  - Expresiones de rango: **a[2-5]z**
  - Cualquier carácter individual: **.**
  - Inicio(**^**) y final (**$**) de línea.
  - Repetición (*****): 0 o más repeticiones.
    - Se puede combinar con **.** para indica cualquier carácter (**.***)
  - Escape (****).
- Expresiones regulares extendidas:
  - Operadores de repetición adicionales.
    - **+ :** 1 o más repeticiones.
    - **?** : 0 o 1 repetición.
  - Múltiples cadenas posibles: **car | truck**
  - Paréntesis: delimitan subexpresiones.

Ficheros – búsqueda: mandato find

- `find`: busca ficheros y directorios recursivamente.
  - Muchas opciones de búsqueda.
  - Trabaja sobre atributos de fichero.
  - Ejemplos:
    ```
    find /usr/lib -name "ls*.so"
    find /home -name "*.txt" -size +1000k
    ```

Option	Description
-name pattern	You can search for files using their names with this option. Doing so finds files that match the specified pattern. This pattern is not technically a regular expression, but it does support many regular expression features.
-perm mode	If you need to find files that have certain permissions, you can do so by using the -perm mode expression. The mode may be expressed either symbolically or in octal form. If you precede mode with a +, find locates files in which any of the specified permission bits are set. If you precede mode with a -, find locates files in which all the specified permission bits are set.
-size n	You can search for files based on size with this expression. Normally, n is specified in 512-byte blocks, but you can modify this by trailing the value with a letter code, such as c for characters (bytes) or k for kilobytes.
-group name	This option searches for files that belong to the specified group.
-user name	This option searches for files that are owned by the specified user.
-maxdepth levels	If you want to search a directory and, perhaps, some limited number of subdirectories, you can use this expression to limit the search.

Ficheros – permisos

- Permisos: los permisos regulan quién puede acceder a un fichero y de qué forma.
  - root: usuario administrador de la máquina.
- Cada fichero tiene permisos para:
  - El dueño del fichero.
  - Para el grupo.
  - Para el resto de usuarios.
- Existen 3 tipos distintos de permisos:
  - Lectura (**r**).
  - Escritura (**w**).
  - Ejecución (**x**).

Ficheros – permisos

- El mandato **ls** muestra a la izquierda los permisos de los ficheros cuando se invoca con la opción de formato largo (**-l**).

```
$ ls -l
-rwxr--r-- 1 user users 6750 5 feb 13:56 archivo1
lrwxrwxrwx 1 user group 8 1 mar 17:26 enlace1 - archivo1
drwx------ 2 user group 4096 12 abr 11:05 directorio
```

  - El primer carácter indica propiedades especiales del fichero: **d** directorio, **l** enlace simbólico, etc.
  - Los siguientes 9 caracteres indican de 3 en 3 los permisos de lectura (**r**), escritura (**w**) y ejecución (**x**) para el dueño, grupo y el resto del mundo.

Ficheros – permisos

- Casos especiales:
  - Directorios:
    - **`r`**: listar directorio.
    - **`w`**: crear una nueva entrada en el directorio.
    - **`x`**: posibilidad de atravesar el directorio.
  - Enlaces simbólicos. Tienen todos los permisos activos ya quien controla los permisos es el fichero enlazado
  - El usuario **`root`** no necesita permisos para acceder a los ficheros, aunque sí necesita el permiso de ejecución para ejecutarlos.

Ficheros – permisos

- Sólo el propietario del fichero puede cambiar los permisos.
- Para cambiar los permisos de un archivo se utiliza el mandato **chmod**.
- Uso:

  chmod [ugoa][+-=][rwx] nombre_fichero

  - **[ugoa]** indica el permiso de quién se va a cambiar: **u**ser, **g**roup, **o**thers, **a**ll.
  - **[+-=]** indica si se añade (**+**), si se quita (**-**) o si se asigna (**=**).
  - **[rwx]** indica los permisos que se modifican **r**ead, **w**rite, e**x**ecute.
  - Se puede especificar una máscara numérica (en octal) para asignar todos los permisos (equivalente a asignarlos con **=**).
- **chown**: cambia el usuario y grupo de archivos y directorios.
  - Ejemplo: **chown user.group nombre_fichero**

Ficheros – permisos

- Ejemplos:
  - Dar al usuario permiso de ejecución:

    `chmod u+x nombre_fichero`
  - Quitar al resto de usuarios permiso de escritura:

    `chmod o-w nombre_fichero`
  - Dar al usuario y al grupo permisos de escritura:

    `chmod ug+w nombre_fichero`
  - Quitar a todos permisos de escritura y ejecución:

    `chmod a-wx nombre_fichero`
  - Especificar todos los permisos:

    `chmod 740 nombre_fichero`

Ficheros – Modificar atributos

- Mandato "umask": Establece los permisos por defecto para los nuevos archivos y directorios creados por el proceso actual
  - Obs: La mayoría de los sistemas no permiten que nuevos archivos sean creados con permisos de ejecución activados
- Se puede especificar:
  - Máscara simbólica: indica los permisos que SÍ se establecerán.

    $ umask u=rwx,g=rwx,o=
  - Máscara numérica (en octal): controla qué permisos son NO se establecerán resultado de una resta bitwise)

    $ umask 0174

    $ mkdir foo

    $ touch bar

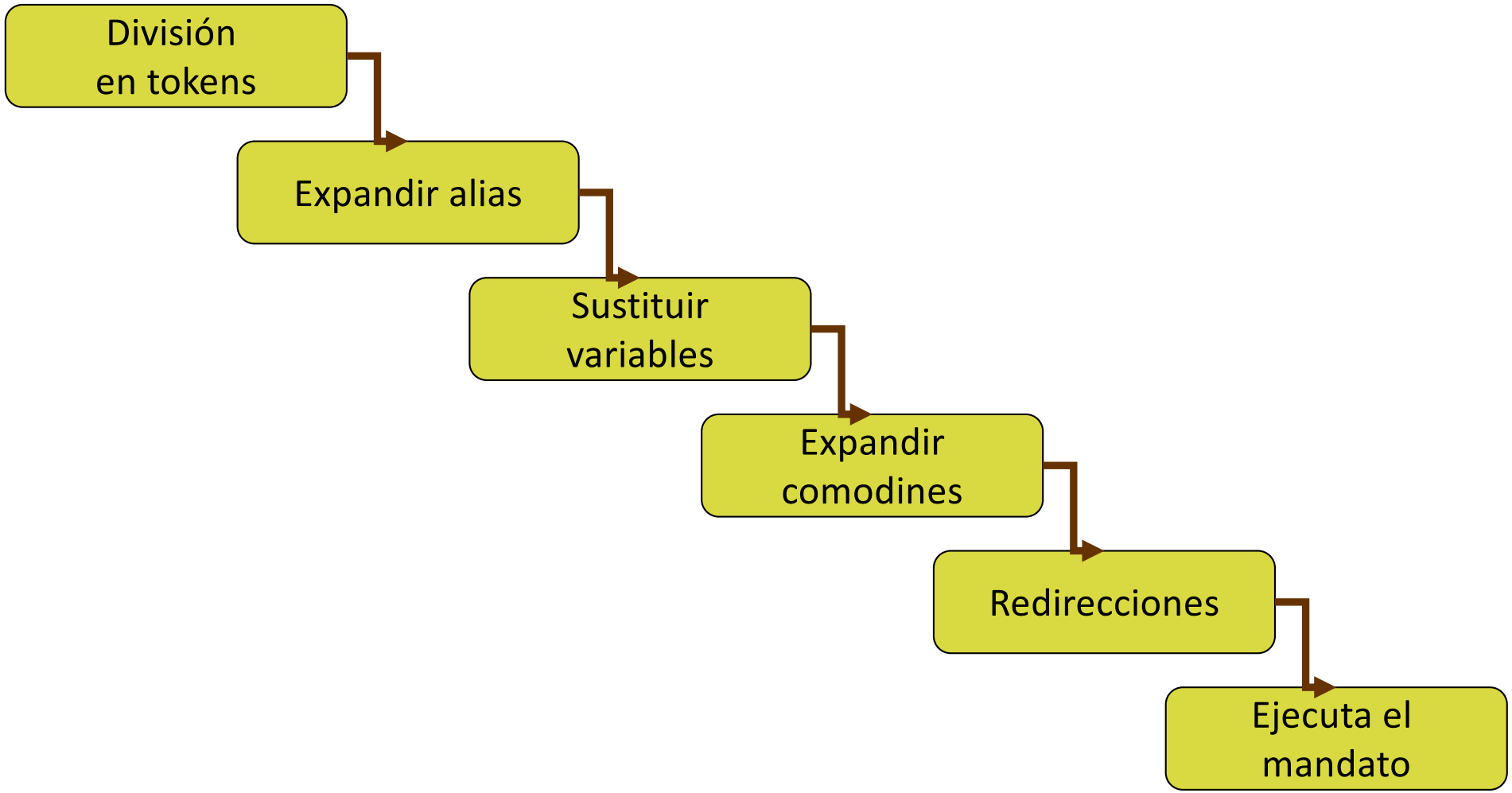
    $ ls -l

    drw-----wx 2 dave 512 Sep 1 20:59 foo

    -rw-----w- 1 dave   0 Sep 1 20:59 bar

Índice

Evaluación de un mandato



Alias de mandatos

- En todas las *shell* se pueden establecer “sobrenombres”.
  - Pueden incluir parámetros.
  - Sintaxis dependiente de la *shell*.
  - Por ejemplo:

```
alias ll='ls -l'
alias rm='rm -i'
```

- Ver los “sobrenombres” creados:

```
alias
```

- Eliminar un “sobrenombre”:

```
unalias ll
```

Variables

- En el entorno de la *shell* se pueden declarar variables dándoles valor directamente:

```
NOMBRE=valor
```

- Se suele utilizar mayúsculas para el nombre de la variable.
- Para usar la variable se utiliza **`$NOMBRE`** o **`${NOMBRE}`** en cualquier parte de una línea:

```
user$ EDAD=16 ; echo $EDAD
16
```

Variables

- Cuando se crea un proceso se le pasa un conjunto de cadenas de caracteres del tipo **VARIABLE=valor**
- Hay variables predefinidas como:
  - **HOME**: el directorio inicial del usuario.
  - **HOSTNAME**: nombre de la máquina.
  - **PATH**: lista de directorios donde la *shell* busca ejecutables.
    - **which**: mandato que busca ficheros ejecutables dentro del **PATH**.
- Los mandatos **env** y **printenv** muestran las variables definidas (sólo si son exportables a los procesos hijos).
  - Ver individualmente:
    - **echo $VAR** *(muestra cualquier variable)*
    - **printenv VAR** *(sólo muestra variables exportables)*
  - Asignar valor:
    - **export VAR=/usr/bin** *(la variable se exportará a nuevos procesos)*
    - **VAR=/usr/bin** *(la variable NO se exportará a nuevos procesos)*
  - Quitar valor:
    - **unset VAR** *(elimina la variable)*

Operaciones con variables

- Las variables almacenan información alfanumérica.
  - Sin embargo, todas las variables y operadores se tratan como *strings*.
- Para realizar operaciones aritméticas se debe utilizar los mandatos **bc** o **expr**, o el **$((operacion))**:

```
user$ expr 5 + 3
8
user$ echo “2 *8” | bc
16
user$ echo $((7*3))
21
```

Sustitución de nombres de fichero

- Existen caracteres comodín que sirven para referirse a un conjunto de ficheros simultáneamente:
  - ***** : cualquier nombre de longitud 0 o más.
  - **?** : cualquier carácter.
  - **[absfg]** : cualquier carácter de la lista.
  - **[!absfg]** : cualquier carácter que no esté en la lista.
  - **[b-n]** : cualquier carácter del rango.
- Observación: Linux es *case-sensitive* (distingue entre mayúsculas y minúsculas) frente a Windows que no lo es.
- Ejemplos:
  - `*.dat` : cualquier archivo cuyo nombre acabe en "**.dat**"
  - `practica[0-3].c` : cualquier archivo que se llame practica, con un número del 0 al 3 y con extensión "**.c**"

Sustitución de mandato

- Objetivo: ejecutar un mandato antes de que se evalúe la línea completa.
- Se consigue encerrando el mandato entre comillas invertidas `` `mandato` `` o con `$(mandato)`.
- Ejemplo:

```
user$ echo Mi login es `whoami` y hoy es $(date)
Mi login es user hoy es Mon Jan 21 12:55:40 CET 2011
```

Escape

- El caracter de escape es ****
- Tiene dos utilidades:
  - Dividir una orden en varias líneas:

```
user$ mandato1  parametro1 parametro2 \
parametro3
```

  - Imprimir caracteres especiales como **$, “** ó ****:

```
user$ echo \$HOME = $HOME
$HOME = /home/user
```

Personalización de la shell

- Cada *shell* tiene sus propios ficheros de configuración:
  - bash: **.bashrc**
  - tcsh y csh: **.cshrc**
  - ksh: **.kshrc**

- Todos los *shell* permiten definir:
  - Alias.
  - Variables de entorno.
  - Path.
  - Variables de configuración de la propia *shell*.

Índice

- Introducción.
- Control de la ejecución.
- Ficheros.
- Sustituciones.
- **Redirecciones**.
- Scripts.

Redirección de entrada y salida

- La entrada estándar (*stdin*) es el sitio de donde lee los datos de entrada un mandato o programa, normalmente es el teclado (teclado habitualmente).
- La salida estándar (*stdout*) es el sitio donde escribe mensajes un mandato o programa, normalmente es la pantalla (monitor habitualmente).
- También existe una salida de error (*stderr*) que es donde envían los mandatos o programas los mensajes de error (monitor habitualmente).
  - Objetivo: que los mensajes de error no se confundan con los de la salida habitual.
  - Se puede redirigir a **/dev/null** si no interesa.

Redirección de entrada y salida

- Se puede redireccionar *stdin, stdout* y *stderr* de cualquier mandato o programa:
  - `mandato < fichero`: coge *stdin* del fichero.
  - `mandato > fichero`: envía *stdout* al fichero (crea el fichero, o si ya existe lo sobreescribe).
  - `mandato >> fichero`: envía *stdout* al fichero (crea el fichero, o si ya existe añade lo nuevo al final).
  - `mandato 2> fichero`: envía *stderr* al fichero.
  - `mandato 2>&1`: envía *stderr* al mismo sitio donde vaya *stdout*.
  - `mandato &> fichero`: envía *stderr* y *stdout* al fichero.

Redirección de entrada y salida

- Ejemplos:
  - Redirigir la entrada estándar

    wc –l Espera a que introduzcamos datos por la entrada estándar (Ctrl - D)

    wc –l < /etc/fstab Redirige la entrada estándar para que sea un fichero

  - Redirigir la salida estándar

    ls Devuelve el resultado en la pantalla

    ls > sal.txt Redirige la salida estándar a un fichero (si existe lo trunca)

    ls >> sal.txt Redirige la salida a un fichero (si existe lo añade)

  - Redirigir la salida de error

    ls fichQueNoExiste 2> sal.txt Redirige la salida de error

  - Redirigir la salida de error a la salida estándar

    ls 2>&1 > sal.txt

Redirección usando tuberías (pipes)

- Otra forma de redirección son los *pipes* (**tuberías**), que permiten redireccionar el *stdout* de un mandato al *stdin* de otro mandato:

  mandato1* | *mandato2* | … | *mandatoN

  com1* < fich_ent | *com2* | … | *comN* > *fich_sal

- Ejemplo:

```
$ ls -la | wc -l
31
$ ls -l /etc | grep user
-rw-r--r--1 root root 981 2010-08-16 11:34 adduser.conf
-rw-r--r--1 root root 600 2010-01-27 11:26 deluser.conf
```

Redirección usando tuberías (pipes)

- No todos los programas reciben su entrada por la entrada estándar.
  - Pueden recibirlos por argumentos.
- Para unir programas que generan su salida por *stdout* con otros que reciben su entrada por argumentos se puede:
  - Ejecutar el mandato antes de que se evalúe la línea completa.
    - Se consigue encerrando el mandato entre comillas invertidas `` `mandato` `` o con `$(mandato)`
      - `rm `find ./ -name "*~"``
  - Usar `xargs`
    - `find ./ -name "*~" | xargs rm`
    - `xargs -n1 mandatom` por cada línea recibida llama al mandato

Índice

- Introducción.
- Control de la ejecución.
- Ficheros.
- Sustituciones.
- Redirecciones.
- **Scripts.**

Scripts

- La *shell* tienen un lenguaje de *scritpting* que permite escribir programas utilizando los mandatos y algunas estructuras de control de flujo.
- Para separar sentencias se utiliza el cambio de línea o el **;**
- El carácter **#** se utiliza para comentarios.
- La primera línea de un *shell script* indica qué *shell* debe utilizarse para interpretar el *script.*
- Ejemplo:

```
#!/bin/bash
echo "Hola Mundo desde $HOSTNAME"
echo "Estamos a `date`"
```

Scripts – ejecución

- Al ejecutar un *script* se crea una *subshell* que lo interpreta y ejecuta.
- Un *script* se ejecuta igual que un mandato cualquiera, invocando su nombre y **siempre teniendo permiso de ejecución** sobre el fichero:
  - *Script* que **SÍ** está en una ruta del **PATH**:
    ```
    $ nombre_script
    ```
  - *Script* que **NO** está en una ruta del **PATH**:
    ```
    $ ruta/nombre_script
    ```
    ruta: parcial o completa
  - *Script* que está en el directorio actual (y no en el **PATH**):
    ```
    $ ./nombre_script
    ```
    antes: `chmod u+x nombre_script`
- Se puede ejecutar un *script* con o sin permiso de ejecución invocando de forma explícita a la *shell*:
  ```
  $ bash nombre_script
  ```
  se puede dar una ruta parcial o completa
- Se puede ejecutar un *script* en el entorno actual sin crear una *subshell* (no es necesario que el *script* tenga permiso de ejecución):
  ```
  $ source nombre_script
  $ . nombre_script
  ```
  en ambos casos se puede indicar una ruta parcial o completa

Scripts – parámetros

- Los *scripts* pueden recibir una serie de parámetros o argumentos cuando se ejecutan.
- Para acceder a los parámetros dentro del *script* existen unas variables especiales:
  - **$#** : número de parámetros.
  - **$n** : argumento n-ésimo ($1 primer argumento, $2 segundo argumento, etc).
  - **$*** : todos los argumentos en una única cadena.
  - **$@** : todos los argumentos, cada uno en una cadena.
  - **$?** : código de error del último mandato ejecutado.
  - **$$** : **pid** de la *shell* que ejecuta el *script*.
- Ejemplo:

```
#!/bin/bash
echo "Se han recibido $# argumentos"
echo "Los argumentos son $*“
```

Scripts – if y test

- Estructura condicional:

```
if [ conditional-expression ]
   then
      commands
   else
      other-commands
fi
```

- No se pueden hacer comparaciones del tipo a > b directamente. Es necesario el mandato **`test`**.
- **`test`** permite hacer comparaciones y otras comprobaciones.
  - **`test x –eq y`** : comprueba si el número **`x`** es igual a **`y`**.
  - Otras comparaciones numéricas: **`–ne, -lt, -le, -gt, –ge`**
  - Condicionales: **&&** (and), **||** (or)
- Una forma alternativa a **`test x –eq y`** es **`[ x –eq y ]`**

Scripts – if y test

- También se puede comparar cadenas:
  - **`test -z cadena`**: comprueba si la cadena está vacía.
  - **`test -n cadena`**: comprueba si la cadena NO está vacía.
  - **`test cadena1 = cadena2`**: comprueba si las cadenas son iguales.
- También se puede hacer comprobaciones sobre ficheros:
  - **`test -e fichero`**: comprueba si existe el fichero.
  - **`test -f fichero`**: comprueba si es un fichero.
  - **`test -s fichero:`** comprueba si existe y contiene datos.
  - **`test -d fichero`**: comprueba si es un directorio.

Scripts – if y test

```
#!/bin/bash
if test $# -ne 2
then
   echo "Numero de argumentos incorrecto. Debe ser = 2“ ; exit 1 ;
fi
if ! test -e $1
then
   echo "El fichero $1 no existe“ ; exit 1 ;
fi
if test -e $2
then
   echo "El fichero $2 existe. ¿Sobreescribir? (s/n)"
   read  OPCION
   if test $OPCION = s
   then
        echo "Sobreescribiendo"
        cp $1 $2; exit $? ;
   else
        echo "Saliendo sin sobreescribir“ ; exit 1 ;
   fi
else
   cp $1 $2; exit $? ;
fi
```

Scripts – case

```
case word in
   pattern1) command(s) ;;
   pattern2) command(s) ;;
   ...
esac
```

Scripts – for

- El bucle **for** tiene una sintaxis distinta a la habitual de otros lenguajes:

```
for var
do
. . .
done
```

Recorre una vez el bucle por cada parámetro que recibe el *script*, tomando **var** el valor de un parámetro en cada pasada.

- Ejemplo que imprime todos los parámetros recibidos:

```
#!/bin/bash
for i
do
   echo "Recibido parametro $i“
done
```

Scripts – for

- El bucle **for** también puede recorrer los elementos de una lista:

```
for var in lista
do
. . .
done
```

Recorre una vez el bucle por cada valor en lista, tomando **var** dicho valor. Las listas se pueden dar de muchas formas:

- Nombres de archivo: ***.txt**
- Valores numéricos: {1..N} ⇒ **`seq 1 N`**
- En general, se puede utilizar cualquier mandato que devuelva una lista de números o palabras.

- Ejemplo: para cada archivo con extensión **.txt** crea un nuevo archivo añadiendo la extensión **.bak**

```
#!/bin/bash
for j in *.txt
do
   cp $j $j".bak"
done
```

Scripts – bucles while y until

- El bucle **`while`** se ejecuta mientras una condición es verdadera:

```
while condicion
do
…
done
```

- Las condiciones se pueden construir con el mandato **`test`.**

- **`until`** es similar a **`while`**, pero se ejecuta mientras la condición es falsa.

Scripts – funciones

- No se necesita definir argumentos. Se pasan como tal.
- Se puede poner **`function`** o no.
- **`return`** saca de la función y continúa la ejecución del *script*.
- Ojo: **`exit`** finaliza programa.
  - 0 correcto.
  - 1-255 incorrecto.
- Dentro de la función **`$#`**, **`$1`**, **`$2`**, **`$3`** … se refieren a los parámetros de la función.

```
#/bin/bash

doit() {
   cp $1 $2
   return;
}

function check() {
  if [ -s $2 ]
     then
        echo "Target file exists!
Exiting!"
        exit 1
  fi
}

check $1 $2
doit $1 $2
exit 0
```