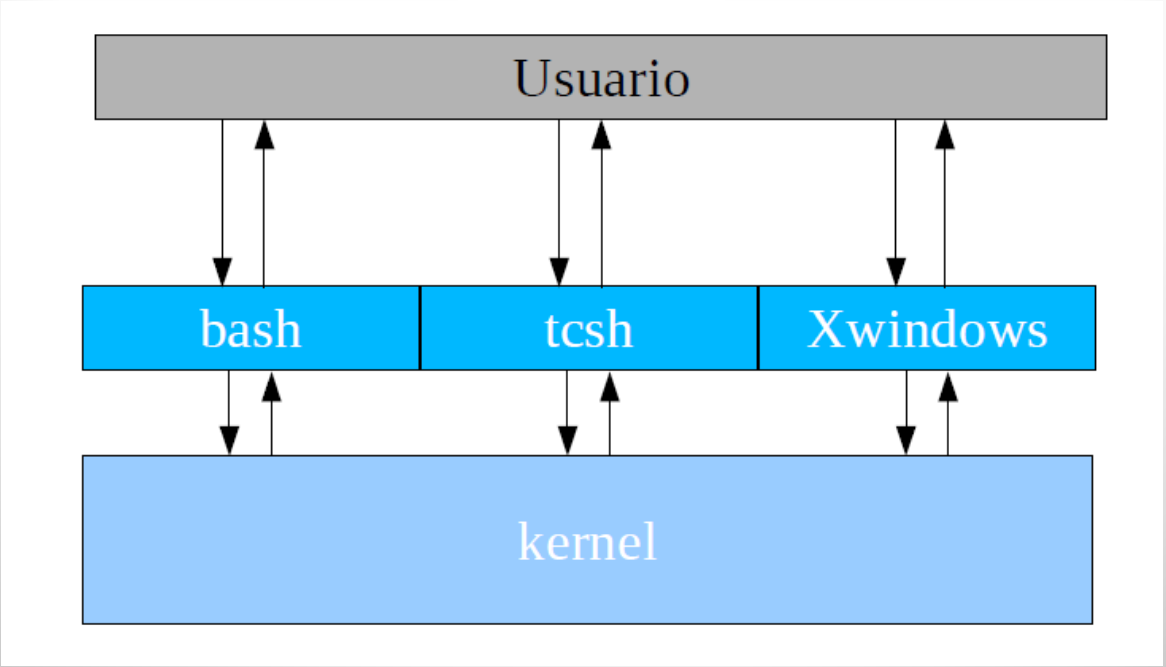
Linux Shell
Command Line Interface

Concepto de Shell

- Programa que actúa como interfaz entre el usuario y el sistema operativo. Permite a aquel introducir órdenes para que sean ejecutadas por éste.



Shell: Command Line Interface

- Existen dos modos posibles de trabajar con una Shell. La primera es mostrando un cursor o **prompt.**
- El usuario escribe directamente las órdenes a través de la línea de comandos, de tal manera que son interpretadas y ejecutadas inmediatamente por el sistema operativo.

Shell: Command Line Interface

- Se implementa un bucle para poder ejecutar las órdenes de forma inmediata, según los siguientes pasos:

```
Cursor -> Introducir comando -> Interpretar comando
-> Ejecutar comando -> Salida de comando -> Cursor -> ...
```

Shell: Scripting

- La segunda forma es lanzando los comandos a través de un guión o **script**.
- Un guión es un fichero de texto cuyo contenido son órdenes del sistema operativo y estructuras de control similares a las usadas en lenguajes con programación estructurada como Pascal o C.
- Asignando permisos de ejecución a este fichero, podemos lanzarlo como un comando más, ejecutando estos comandos en forma secuencial, o con bifurcaciones condicionales y bucles.

Shell Bash

- Existen distintas variantes de intérpretes shell, entre otras: Bourne (sh), Korn (ksh93), Bash (bash), Z-Shell (zsh), C-Shell (csh), TC-Shell (tcsh).
- Nosotros utilizaremos la shell Bash. El nombre proviene de Bourne Again Shell, y nació como uno de los proyectos GNU de la FSF.
- La filosofía general de FSF es desarrollar software completamente gratuito y de libre distribución, bajo licencia GNU.
- Podemos comprobar si están instaladas con la orden ***which*** nombreprograma

Shell Bash

- Bash es una de las shells más usadas hoy en día, principalmente porque ha sido adoptada como shell predeterminada en las distribuciones de Linux.
- La característica de ser un proyecto GNU le da el atractivo de ser totalmente libre, por lo que también puede ser usada en cualquier otra distribución UNIX; sólo es necesario bajar el código y compilarlo, o buscar el ejecutable ya compilado para nuestro sistema.

Órdenes externas

- Cuando tecleamos una orden para que sea ejecutada por la shell, generalmente ésta busca en los directorios listados en la variable de entorno **$PATH** un fichero ejecutable con el nombre de la orden en cuestión (a no ser que se especifique la ruta completa a la orden dentro del sistema de ficheros)
- Estas son las llamadas órdenes externas

Órdenes externas

```
$ echo $PATH
/home/juan/bin:/usr/local/bin:/usr/bin:/bin (lista de directorios separados por ':')
$ ls (encuentra el ejecutable /bin/ls)
$ ./miscript (especificamos la ubicación del ejecutable)
```

Órdenes internas

- La shell dispone, sin embargo, de una serie de órdenes internas (*builtins*) que puede ejecutar directamente sin necesidad de buscar el ejecutable, pues ya tiene incluida esa funcionalidad en su propio código.

Órdenes internas

- Estas son algunas habituales:
  - cd [dir]
  - exit [valor]
  - history
  - fg
  - bg
  - jobs

Foreground vs. background

- Normalmente, las órdenes externas se ejecutan de forma interactiva (***foreground***): la shell interpreta la orden, la ejecuta, nos muestra la salida de la orden, y no nos devuelve el prompt hasta que no acaba de ejecutarse el proceso.
- Cuando queremos dejar un programa en ejecución de manera no interactiva o en segundo plano (***background***) se emplea el operador &.

Foreground vs. background

- Ejemplo:

```
$ xlogo &
[2] 7584
$
```

- El primer número, en este caso el [2], es el número de tarea (job), mientras que el segundo número (7584) es el identificador de proceso, PID.
- Una vez que el proceso ha iniciado la ejecución como un proceso separado, ya no podemos interrumpirlo mediante la combinación Ctrl-C.

Foreground vs. background

- La orden ***bg*** (de background) nos permite pasar a segundo plano un proceso que se encuentra suspendido, por ejemplo porque se le ha enviado la señal correspondiente mediante la combinación de teclas Ctrl+Z.
- La orden ***fg [tid]*** permite traer a primer plano (foreground) un proceso que se encontraba en background.

Foreground vs. background

- Para consultar cuáles son los procesos que se están ejecutando actualmente en segundo plano puede emplearse la orden ***jobs***.
- Cuando un proceso que se estaba ejecutando en background finaliza, la shell muestra un mensaje indicando tal circunstancia.

Foreground vs. background

- Para terminar la ejecución de un proceso en background hemos de enviarle la señal de terminación (SIGTERM) mediante la orden ***kill***, indicando como parámetro bien el número de PID, bien el número de tarea precedido por el carácter %.
- Si no responde a SIGTERM, podemos enviar SIGKILL.

Ejercicio

- Lanzar 4 procesos xeyes en background
- Traer a foreground y detener los 2 procesos creados en segundo y cuarto lugar
- Terminar los 2 procesos que siguen en ejecución usando la orden kill
- Traer los 2 restantes a foreground (uno a uno) y eliminarlos con Ctrl-C

Variables y aritmética

- Bash permite la definición de variables cuyo valor puede ser recuperado más tarde. Por ejemplo:

```
$ ancho=24
$ echo $ancho
24
$ saludo="Hola, ¿cómo estás?"
$ echo $saludo
Hola, ¿cómo estás?
$
```

Variables y aritmética

- Al asignar un valor a una variable no deben aparecer espacios alrededor del símbolo =
- Resulta un error en caso contrario, pues la shell considera que el nombre de la variable es una orden que debe ejecutar con una serie de argumentos:

```
$ ancho = 24
bash: ancho: command not found
```

Variables y aritmética

- Se pueden realizar operaciones aritméticas sobre variables mediante el operador $((expresión)). Por ejemplo:

```
$ ancho=24
$ echo $ancho
24
$ ancho=$(( $ancho+1 ))
$ echo $ancho
25
```

- Pueden usarse operadores aritméticos y de bits: +, -, /, *, %, &, |, ^, <<, >>

Variables y aritmética

- Podemos convertir una variable en variable de entorno mediante la orden ***export***
- Las variables de Shell están disponibles sólo para la Shell en sí, las variables de entorno están disponibles para cualquier programa lanzado desde la Shell
- Podemos ver las variables de entorno con la orden ***printenv***

Ejercicio

- Crear una variable con nombre a y valor 5
- Crear una variable b con valor a+3
- Crear una variable c con valor b desplazado 3 bits a la izquierda
- Imprimir los valores de a, b y c

Redirecciones

- Todo proceso Unix tiene asociados de manera predeterminada 3 descriptores de ficheros: la entrada estándar (0), la salida estándar (1) y la salida de error (2).
- A la salida de error se dirigen los mensajes provocados por algún error en la ejecución del proceso.

Redirecciones

- La salida estándar de una orden puede redirigirse a un fichero mediante el operador >fichero. Por ejemplo: $ ls – l /etc >dir_etc.txt
- Con el operador >fichero se trunca el fichero designado. Si queremos añadir el resultado de una orden al contenido de un fichero existente, debemos utilizar el operador de doble redirección >>fichero

Redirecciones

- También se puede redirigir la salida de error mediante el operador de redirección 2>fichero. Por ejemplo:

```
$ ls -l /bin/bash /bin/noexiste
ls: no se puede acceder a /bin/noexiste: No existe el fichero o el directorio (salida de error)
-rwxr-xr-x 1 root root 700492 may 12 2008 /bin/bash (salida estándar)
$ ls -l /bin/bash /bin/noexiste >/dev/null (se redirige la salida estándar)
ls: no se puede acceder a /bin/noexiste: No existe el fichero o el directorio (salida de error)
$ ls -l /bin/bash /bin/noexiste 2>/dev/null (se redirige la salida de error)
-rwxr-xr-x 1 root root 700492 may 12 2008 /bin/bash (salida estándar)
```

Redirecciones

- Vemos que podemos redirigir la salida de error indicando el número de descriptor de fichero justo antes del operador de redirección.
- También podemos redirigir la salida de error a la salida estándar mediante el operador >&, para que al redirigir esta última un fichero aparezcan redirigidos tanto los mensajes de error como la salida normal.
- **Ejercicio**: repetir el ejemplo anterior, pero volcando tanto la salida estándar como la de error a un fichero llamado salidas.txt

Cauces (pipes)

- La salida estándar de una orden puede servir también como entrada estándar para la siguiente. Ambas órdenes se encadenan mediante el operador | (cauce o pipe). Por ejemplo:

```
$ ls -l /etc | sort | more
drwxr-s---  2 root   dip     96 dic 15 12:53 chatscripts
drwxr-sr-x  2 root   bind   352 ene 16 15:15 bind
drwxr-xr-x 12 root   root   608 oct 20 11:41 X11
drwxr-xr-x 13 root   root   352 nov 20 11:20 texmf
```

Cauces (pipes)

- Se pueden combinar cauces y redirecciones. Es muy importante tener en cuenta que las redirecciones de salida se realizan antes de ejecutar las órdenes.

Ejercicio

- Crear un fichero llamado datos.txt utilizando el comando cat. Contendrá 5 líneas, cada línea contendrá un número menor que el de la línea anterior.
- Usando el comando sort, redirecciones y/o cauces, crear un fichero datos_ord.txt que contenga las líneas de datos.txt ordenadas de menor a mayor.

Listas de órdenes

- Una lista de órdenes es una secuencia de comandos (con redirección o no) separados por los operadores: ; & && ||
- En una lista de órdenes separadas por el operador ; la shell ejecuta las órdenes de manera secuencial, esperando que se complete cada uno de los comandos. El código de retorno (el valor de la variable $?) es el de la última orden ejecutada.

Listas de órdenes

- Si el operador es &, el comando se ejecuta en segundo plano, la shell no espera a que termine y el código de retorno es 0.
- Los operadores && y || son operadores AND y OR respectivamente.
- orden1 && orden2 sólo se ejecuta orden2 si orden1 retornó con código 0
- orden1 || orden2 sólo se ejecuta orden2 si orden1 retornó con código distinto de 0

Listas de órdenes

- ¿En qué casos de los siguientes sería 0 el código de retorno?

```
$ ls /bin/bash && echo Existe ; echo $?

$ ls /bin/noexiste && echo existe ; echo $?

$ ls /bin/bash || echo No existe ; echo $?

$ ls /bin/noexiste || echo No existe ; echo $?
```

Expansión de comodines

- A veces queremos referirnos a un conjunto de ficheros que tienen características comunes en sus nombres.
- Si queremos mostrar en pantalla (usando cat) todos los ficheros que contengan código fuente C, la característica común es que los nombres de los ficheros acaban en .c :

```
$ cat *.c
```

Expansión de comodines

- El carácter * concuerda con cualquier cadena de caracteres.
- El carácter ? concuerda con cualquier carácter
- [conjunto] concuerda con cualquier carácter dentro de conjunto.
- ¿Qué ficheros listaría la siguiente orden?
  - ls ?[a-c]*.h

Expansión de órdenes

- Podemos guardar en una variable la salida estándar de una orden o lista de órdenes.
- Esto lo hacemos mediante la expansión de órdenes: $(orden)
- Ejemplo:

```
$ num=$( ls a* | wc -w )
$ echo $num
13
```

Algunas órdenes útiles: grep

- La orden ***grep*** busca en la entrada (bien en la que se le especifica con nombres de ficheros, o bien en la entrada estándar si no se le dan dichos nombres o si uno de éstos consiste en –), líneas que concuerden con el patrón dado.
- Normalmente grep muestra las líneas que concuerden, pero se puede alterar su comportamiento con distintas opciones

Algunas órdenes útiles: wc

- La orden ***wc*** escribe el número de saltos de línea, de palabras o de caracteres en un fichero.
- Si se especifican varios ficheros, se escribe también el total.
- Si no se especifica el fichero, o el nombre es -, entonces se toma la entrada estándar.

Algunas órdenes útiles: sort

- La orden ***sort*** escribe la concatenación ordenada del fichero(s) a la salida estándar.
- Si se omite fichero, se toma la entrada estándar.
- Se reconocen varias opciones de ordenación (ignore-case, reverse, numeric-sort, etc.)

Algunas órdenes útiles: cut

- La orden ***cut*** escribe en la salida estándar partes seleccionadas de cada línea de sus ficheros de entrada.
- Si se omiten los ficheros de entrada, se toma la entrada estándar.
- Tiene opciones para seleccionar distintos delimitadores, listas de caracteres y campos

Ejercicio

- Utilizando varias órdenes en una sola línea, escribir en un fichero llamado alumnos_z.txt el login de todos los usuarios del directorio /home/alumnos
- Utilizando varias órdenes en una sola línea, y leyendo el fichero anterior, escribir a la salida estándar el número de alumnos cuyo login empieza por z

Ejercicio

- Descargar el fichero que contiene el libro de Alicia en el país de las maravillas de Internet (en inglés, formato “plain text”)
- Utilizando varias órdenes en una sola línea, crear un fichero llamado alice_the.txt que contenga el número de líneas del fichero en las que aparece la palabra “the” (no la cadena) al menos una vez
- Repetir lo anterior de forma “case insensitive”

Ejercicio

- Utilizando varias órdenes en una sola línea, guardar el listado de las bibliotecas diferentes que existen en el directorio /usr/lib/x86_64-linux-gnu, independientemente de su extensión y del número de versión, ordenadas por orden alfabético en el fichero listalibs.txt.
- Se recomienda usar el comando uniq para evitar duplicados