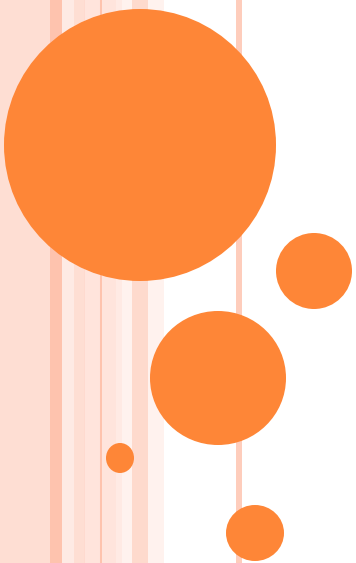


Usaremos sh, las herramientas de bash están prohibidas (a excepción de las que se indiquen en clase )—> SI PONES ALGO EN EL EXÁMEN QUE NO SE HA VISTO EN CALSE SUSPENDES

# ADMINISTRACIÓN DE SISTEMAS



**Interacción y programación para la  
administración de sistemas: Estándar  
IEEE std 1003.1 (Posix): shell y  
herramientas I**

# EL SHELL

- El proceso shell se inicia cuando accedemos a nuestra cuenta. Proporciona:
  - un intérprete de comandos
  - un entorno de programación
- Existen diferentes shells:
  - Bourne (*sh*), Korn (*ksh*), Bourne again (*bash*), Z (*zsh*), C (*csh*), TENEX C (*tcsch*), ...
- Ejecución del shell
  - Al inicio, se ejecuta el shell indicado en el último campo de la línea de usuario en */etc/passwd* (se cambia mediante ***chsh***)
  - Desde la línea de comandos: */bin/bash*
- Salir del shell
  - Comando ***exit***
  - **ctrl-D**

# REFERENCIA

- Bash reference manual:

<http://www.gnu.org/software/bash/manual/bashref.html>

# TIPOS DE COMANDOS EN SHELL

- El shell nos permite ejecutar:
  - Comandos externos, por ejemplo: *ls*, *cat*, *mkdir*, etc.
    - son programas ajenos al shell (usualmente en /bin o /sbin)
    - cuando se lanzan inician un nuevo proceso
    - se buscan en los directorios indicados en la variable PATH
  - Comandos internos (*builtin commands*), por ejemplo: *cd*, *bg*, *alias*, *eval*, *exec*, *pwd*, etc.
    - se ejecutan en el mismo proceso del shell, sin lanzar un nuevo proceso
- En bash: para saber si un comando es externo o interno usar el comando interno ***type***:

```
$ type cd
cd is a shell builtin
$ type cat
cat is /bin/cat
```

# ALGUNOS COMANDOS BÁSICOS

Command	Description
<code>cat file(s)</code>	Display contents of <i>file(s)</i> or standard input if not supplied
<code>cd dir</code>	Change working directory to <i>dir</i>
<code>cp file<sub>1</sub> file<sub>2</sub></code>	Copy <i>file<sub>1</sub></i> to <i>file<sub>2</sub></i>
<code>cp file(s) dir</code>	Copy <i>file(s)</i> into <i>dir</i>
<code>date</code>	Display the date and time
<code>echo args</code>	Display <i>args</i>
<code>ln file<sub>1</sub> file<sub>2</sub></code>	Link <i>file<sub>1</sub></i> to <i>file<sub>2</sub></i>
<code>ln file(s) dir</code>	Link <i>file(s)</i> into <i>dir</i>
<code>ls file(s)</code>	List <i>file(s)</i>
<code>ls dir(s)</code>	List files in <i>dir(s)</i> or in current directory if <i>dir(s)</i> is not specified
<code>mkdir dir(s)</code>	Create directory <i>dir(s)</i>
<code>mv file<sub>1</sub> file<sub>2</sub></code>	Move <i>file<sub>1</sub></i> to <i>file<sub>2</sub></i> (simply rename it if both reference the same directory)
<code>mv file(s) dir</code>	Move <i>file(s)</i> into directory <i>dir</i>
<code>ps</code>	List information about active processes
<code>pwd</code>	Display current working directory path
<code>rm file(s)</code>	Remove <i>files(s)</i>
<code>rmdir dir(s)</code>	Remove empty directory <i>dir(s)</i>
<code>sort file(s)</code>	Sort lines of <i>file(s)</i> or standard input if not supplied
<code>wc file(s)</code>	Count the number of lines, words, and characters in <i>file(s)</i> or standard input if not supplied
<code>who</code>	Display who's logged in

# PROGRAMACIÓN SHELL: SCRIPTS

- Bash (y otros shells) permiten programar scripts, programas **interpretados** orientados a comandos, procesos y ficheros.
  - Realización de programas para automatización de tareas de administración.
  - Modificación de programas shell de aplicaciones y de sistema (de instalación, configuración...).
- Script o programa shell : fichero de texto conteniendo comandos externos e internos, que se ejecutan línea por línea.
- El programa puede contener, además de comandos:
  - Variables, parámetros, estructuras condicionales, bucles, funciones, **comentarios (carácter # hasta final de línea)**

# EJECUCIÓN DE UN SCRIPT

- Un programa se construye:
  - Directamente en la línea de comandos
  - En un fichero
- Ejecución de un programa en un fichero:
  - Ejecutar un shell poniendo como argumento el nombre del script (solo necesita permiso de lectura)  
`$> bash fichero_script`
  - Ejecutar el fichero como un comando (necesita permiso de ejecución)  
`$> chmod u+x fichero_script`  
`$> ./fichero_script`

# EJECUCIÓN DE UN SCRIPT

- Por defecto, el script es ejecutado por un proceso del shell en curso. Si se quiere explicitar un shell de ejecución en particular, se pone en la primera línea del fichero el shell a ejecutar con el formato `#!/path_al_shell` (esto se llama “*shebang*”).

```
#!/bin/bash
```

```
#!/bin/sh
```

```
#!/usr/bin/perl
```

- Del manual de `execve`: `execve()` executes the program pointed to by *filename*. *filename* must be either a binary executable, or a script starting with a line of the form:

```
#! interpreter [optional-arg]
```



# EJECUCIÓN DE UN SCRIPT

- Ejecución con el shell actual
  - Los métodos anteriores arrancan un sub-Shell (un proceso hijo) que lee las órdenes del fichero, las ejecuta y después termina cediendo el control nuevamente al shell original (al proceso padre).
- Existe una forma de decirle al shell actual que lea y ejecute una serie de órdenes sin arrancar un sub-shell:

```
$> . helloworld
```

o bien:

```
$> source helloworld
```

# EL SHELL: CARACTERES ESPECIALES

- Hay una serie de caracteres que el shell reconoce y trata de forma especial:

Carácter	Significado
' " \	cambian la forma en que el shell interpreta los caracteres especiales
&	usado después de un comando, indica que se ejecute en background
< > >> << `	caracteres de redirección
* ? [ ] [! ]	caracteres de sustitución (comodines)
\$	indica una variable del shell
;	usado para separar múltiples comandos en la misma línea

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

El que sustituye es el shell, no el programa invocado

- Los *comodines* (*wildcards*) permiten especificar múltiples ficheros al mismo tiempo.
- Ejemplos:

```
$> ls -l *html
```

```
$> ls fichero23.???
```

```
$> ls fichero[23]1.txt
```

```
$> ls fichero[!x].txt
```

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

Carácter	Corresponde a
*	0 o más caracteres
?	1 carácter
[ ]	uno de los caracteres entre corchetes
[! ] o [^ ]	cualquier carácter que no esté entre corchetes

- [ ]: sustitución por rango

[abc]

[0-9]

[z-f]            OJO !!! ERROR

[a-np-z]\*

- Uso de ! (negación)

[!a-z] -> Cualquier carácter excepto minúsculas

\*[!o] -> Cualquier fichero sin la letra 'o' al final

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

- Asterisco: sustitución de strings

```
gvalles@baal-debian:~/capitulos$ ls
cap1 cap2 cap3 cap4 otro pap1 pap11
gvalles@baal-debian:~/capitulos$ echo *
cap1 cap2 cap3 cap4 otro pap1 pap11
gvalles@baal-debian:~/capitulos$ echo ca*
cap1 cap2 cap3 cap4
```

- Interrogación: sustitución de caracteres individuales

```
gvalles@baal-debian:~/capitulos$ echo ?ap?
cap1 cap2 cap3 cap4 pap1
gvalles@baal-debian:~/capitulos$ echo ?????
pap11
```

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

Command	Description
<code>echo a*</code>	Print the <i>names</i> of the files beginning with <code>a</code>
<code>cat *.c</code>	Print all files ending in <code>.c</code>
<code>rm *.*</code>	Remove all files containing a period
<code>ls x*</code>	List the names of all files beginning with <code>x</code>
<code>rm *</code>	Remove <i>all</i> files in the current directory (Note: Be careful when you use this.)
<code>echo a*b</code>	Print the names of all files beginning with <code>a</code> and ending with <code>b</code>
<code>cp ../programs/* .</code>	Copy all files from <code>../programs</code> into the current directory
<code>ls [a-z]*[!0-9]</code>	List files that begin with a lowercase letter and don't end with a digit

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

- Recordar el tema de los caracteres especiales del shell, ojo a su interacción con los comodines

Carácter	Acción
'	el shell ignora todos los caracteres especiales contenidos entre un par de comillas simples
"	el shell ignora todos los caracteres especiales entre comillas dobles excepto \$, ` y \
\	el shell ignora el carácter especial que sigue a \

- Ejemplos:

```
$ echo /usr/bin/a*  
/usr/bin/a2p /usr/bin/aconnect /usr/bin/acpi .....  
$ echo "/usr/bin/a*"  
/usr/bin/a*
```

# SUSTITUCIÓN DE NOMBRES DE FICHEROS

- **Importante:** El shell hace la sustitución **antes de ejecutar** el comando
- Si la expansión falla, bash deja el argumento como está

```
$ echo /usr/bin/asdf*fkj
/usr/bin/asdf*fkj
$ ls /usr/bin/asdf*fkj
ls: /usr/bin/asdf*fkj: No hay tal fichero o directorio
```
- Los ficheros “ocultos” (que empiezan por .) no se expanden, debemos poner el . de forma explícita



# EL SHELL: CARACTERES ESPECIALES

- **bash** permite eliminar el significado de los caracteres especiales, usando los caracteres especiales: `'`, `"` o `\`

Carácter	Acción
<code>'</code>	el shell ignora todos los caracteres especiales contenidos entre un par de comillas simples
<code>"</code>	el shell ignora todos los caracteres especiales entre comillas dobles excepto <code>\$</code> , <code>`</code> y <code>\</code>
<code>\</code>	el shell ignora el carácter especial que sigue a <code>\</code>

- Ejemplos:

```
$ echo '$PATH'
$PATH
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin
$ echo I\'m Pepe
I'm Pepe
```

# VARIABLES

- Asignar un valor: *nombre variable=valor*

```
$ una_variable=hola
$ un_numero=15
$ nombre="Pepe Pota"
```

  - Los espacios en blanco se tienen en cuenta:
    - usar comillas para incluirlos en la variable
- Acceder a las variables: *\$nombre variable*

```
$ nombre="Pepe Pota"
$ echo $nombre
Pepe Pota
$ comando=ls
$ $comando
20041020    DeadLetters    ioports.txt    news
```
- Utilización de llaves si la variable es parte de una palabra mayor: *\${nombre variable}*

```
$ nombre="Pepe Pota"
$ echo ${nombre}mo
Pepe Potamo
```

# VARIABLES

- Uso de variables:
  - programación scripts shell
  - control del entorno de ejecución del shell (*PATH*, *HOME*, ...)
- Dos tipos:
  - variables locales: visibles solo desde el shell actual. Se pueden mostrar ejecutando `set`
  - variables globales o de entorno: visibles en todos los shells. Se pueden mostrar con `env` o `printenv`
- El nombre de las variables debe:
  - empezar por una letra o `_`
  - seguida por cero o mas letras, números o `_` (sin espacios en blanco)

# VARIABLES DE ENTORNO

- Para ver las variables de entorno, **env** o **printenv**

HOME : directorio base del usuario

SHELL : el programa ejecutable para el shell que se utiliza

UID : el id del usuario en curso

USER, USERNAME : el nombre del usuario

TERM : el tipo de terminal en uso

DISPLAY : la pantalla de X-Windows

PATH : El path de ejecución del usuario

PS1/PS2/...: los “prompts” de comandos

PWD: el directorio actual

MANPATH : el path para las páginas del manual

# SUSTITUCIÓN DE COMANDOS: \$(...)

- `$(comando)` o ``comando``
  - Ejecuta comando y reemplaza `$(comando)` por su salida estándar
  - Comando se ejecuta dentro de un subshell y la salida de la sustitución es la salida standard del comando
  - Ejemplo:  

```
as@as $ dirs=$(ls /)
as@as $ echo "$dirs"
bin
boot
dev
...
var
```