



**UNIVERSIDADE DA CORUÑA**

---

**El entorno de programación Unix**

---

**Fernando Bellas Permuy**

Departamento de Tecnologías de la Información y las Comunicaciones (TIC)

Universidad de A Coruña

<http://www.tic.udc.es/~fbellas>

[fbellas@udc.es](mailto:fbellas@udc.es)

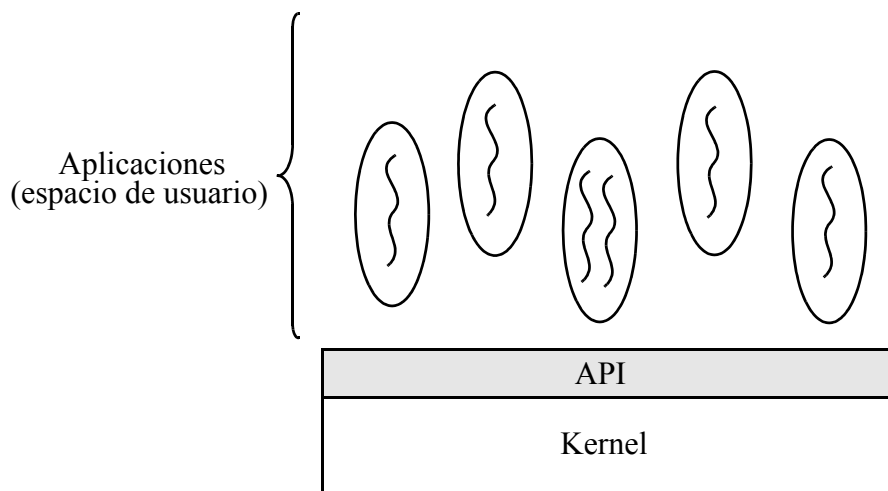
# Contenidos

- El modelo Unix.
- Comandos más usuales.
- La shell.
- Caracteres especiales en Bourne Shell.
- Redireccionamientos de entrada/salida y pipes en Bourne shell.
- Expresiones regulares.
- Programación de scripts en Bourne shell.
- Acceso remoto.
- Arquitectura X Window.
- Arquitectura típica de una red heterogénea Unix/MS-Windows.
- Creación de Makefiles (pendiente para el curso de C++ ...).
- Bibliografía.



# El modelo Unix

- **Proceso**  
Programa en ejecución.
- **Thread**  
Flujo de ejecución dentro de un proceso.
- **Multitarea**  
Una misma máquina ejecuta varios procesos simultáneamente.
- **Multiusuario**  
En una misma máquina se ejecutan los procesos de distintos usuarios.
- **El sistema operativo ofrece sus servicios a través de un conjunto de llamadas al sistema**  
Gestión de ficheros y directorios, creación de procesos, gestión de señales, etc.



- **Distintas versiones de Unix**  
Solaris (Sun), HP-Unix, AIX (IBM), Linux, etc.



# Comandos más usuales (1)

- **ls** (listar)

```
ls -al .
```

```
d rwx --- --x 20 fer      senior      1536 Oct 19 12:11 .
d r-x r-x r-x  2 root     root        3 Oct 19 13:07 ..
- rwx --- ---  1 fer      senior      171 Jul 28 1997 .cshrc
- rwx --- ---  1 fer      senior     6842 Jul 24 1997 .cshrc.old
d rwx --- ---  2 fer      senior      512 Mar 30 1998 Direcciones
d rwx --- ---  2 fer      senior      512 Sep  5 12:27 Personal
d rwx --- --- 15 fer      senior      512 Oct 16 16:55 Trabajos
d rwx --- ---  2 fer      senior      512 Jul 31 22:04 bin
d rwx --- ---  2 fer      senior      512 Oct 16 14:39 dead_letter
- rwx --- ---  1 fer      senior     7487 Sep  1 18:48 fmdictionary
- rw- --- ---  1 fer      senior      257 Oct 19 12:11 fmfilesvisited
d rwx --- ---  8 fer      senior      512 Oct  6 1997 login
d rwx --- ---  2 fer      senior      512 Dec 22 1997 nsmail
d rwx r-x r-x  3 fer      senior      512 Oct 16 17:42 public_html
d rwx --- ---  5 fer      senior     1024 Oct 19 12:24 tmp
```

Otra opción típica: `-R` (recursivo).

- **chmod** (cambiar modo), **chown** (cambiar propietario),  
**chgrp** (cambiar grupo)

```
chmod 744 p.cc
chmod u=rwx,go=r p.cc
```

```
chmod ugo+r p.cc
chmod a+r p.cc
```

```
chmod a-r p.cc
```

```
chown fer p.cc g.cc
chgrp senior p.cc g.cc
```

Todos admiten la opción `-R` (recursivo).



## Comandos más usuales (2)

- **cd** (cambiar directorio), **pwd** (imprimir directorio de trabajo)

```
cd /home/fer/Trabajos/GRACE
pwd
cd Trabajos/GRACE
cd
```

- **cp** (copiar)

```
cp nombreFichero1 nombreFichero2
cp nombreFichero nombreDirectorio
cp nombreFichero1 .. nombreFicheroN NombreDirectorio
cp -r nombreDirectorio1 nombreDirectorio2
cp -r nombreDirectorio1/* nombreDirectorio2
cp nombreDirectorio1/* nombreDirectorio2
```

- **mv** (mover)

```
mv nombreFichero1 nombreFichero2
mv nombreFichero nombreDirectorio
mv nombreFichero1 .. nombreFicheroN NombreDirectorio
mv nombreDirectorio1 nombreDirectorio2
```

- **mkdir** (crear directorio)

```
mkdir nombreDirectorio
```

- **rm** (borrar)

```
rm nombreFichero1 .. nombreFichero2
rm -r nombreDirectorio
rm -rf nombreDirectorio
```

- **rmdir** (borrar directorio)

```
rmdir nombreDirectorio
```

- **cat** (listar fichero), **more** (listar fichero con pausas), **head** (primeras líneas de un fichero), **tail** (últimas líneas de un fichero).

```
cat nombreFichero
more nombreFichero
head nombreFichero
tail nombreFichero
```



## Comandos más usuales (3)

- **diff** (comparación de ficheros de texto), **cmp** (comparación de ficheros binarios)

```
diff ficheroDeTexto1 ficheroDeTexto2
cmp ficheroBinario1 ficheroBinario2
```

- **lpr** (imprimir fichero), **lpq** (comprobar la cola de impresión)

```
lpr ficheroDeTexto1 ... ficheroDeTextoN
lpr -P nombreImpresora ficheroDeTexto1 ... ficheroDeTextoN
lpq -P nombreImpresora
```

- **who** (ver usuarios activos en una máquina), **whoami**, **date** (fecha y hora), **uname** (información sobre el sistema)
- **su** (acceder como otro usuario)

```
su - fbellas
```

- **passwd** (cambiar password)
- **file** (determinar el tipo de fichero)
- **ps** (consultar la tabla de procesos)

```
ps
ps -fea
```

- **kill** (enviar una señal a un proceso)

```
kill -<número de señal> <número de proceso>
kill -9 123
```

- **man** (consultar el manual), **xman** (versión X Window de man)

```
man nombreComando
man -s <número de sección> nombreComando
man kill
man -s 2 kill
```

- **df** (listar espacio en sistemas de ficheros)

```
df -k
```



## Comandos más usuales (y 4)

- **ln** (establecer links)

```
ln nombreFichero nombreLink (hard link)
ln -s nombreFichero nombreLink (symbolik link)
ln -s NombreDirectorio nombreLink (symbolik link)
```

- **wc** (cuenta de líneas, caracteres y palabras)

```
wc fichero
wc -l fichero
wc -l -w fichero
wc -lw fichero
wc fichero1 fichero2 ... ficheroN
```

Opciones: -l y -w.

- **vi** (edición de ficheros)

- **gzip, gunzip** (compresión y descompresión de ficheros - GNU)

```
gzip nombreFichero1 ... nombreFicheroN
gunzip nombreFichero1 ... nombreFicheroN
```

Otra opción -9 para gzip.

- **tar** (empaquetamiento de ficheros)

```
cd
tar cvf tmp.tar /home/fer/tmp
tar tvf tmp.tar
tar cvf tmp.tar tmp
tar tvf tmp.tar
tar xvf tmp.tar
```



# La shell (1)

- **Concepto**

Lee una línea de comandos por la entrada estándar (hasta fin de línea), la analiza, y ejecuta los comandos correspondientes (lanzando procesos hijo).

- **Distintos tipos de shell**

- `/bin/sh` (Bourne Shell), `/bin/ksh` (Korn Shell), `/bin/csh` (C Shell), `bash` (versión interactiva de `sh`), `tcsh` (versión interactiva de `csh`).
- Cada usuario tiene asociado un tipo de shell.

- **Variables de entorno**

- Todo proceso tiene asociadas variables de entorno. Una variable de entorno asocia un nombre (`HOME`) con un valor (`/home/fer`).
- Si el proceso padre lo desea, el proceso hijo hereda todas las variables de entorno del padre.
- Especificación de variables de entorno.
  - `sh, ksh, bash.`
    - `NOMBRE_VARIABLE=valor`
    - Para exportarla => `export NOMBRE_VARIABLE`
  - `csh, tcsh.`
    - `setenv NOMBRE_VARIABLE valor`

- **Ficheros de inicialización/finalización**

- `sh, ksh, bash:`
  - `/etc/profile` (la primera vez que un usuario entra en su cuenta).
  - `$HOME/.profile` (la primera vez que un usuario entra en su cuenta).
- `csh, tcsh:`
  - `/etc/.login` (la primera vez que un usuario entra en su cuenta).
  - `$HOME/.login` (la primera vez que un usuario entra en su cuenta).
  - `$HOME/.logout` (se ejecuta al terminar la sesión).
  - `$HOME/.cshrc` (cada vez que se ejecuta un shell).
- Los ficheros de inicialización/finalización son scripts.





## La shell (2)

- **Ficheros de inicialización/finalización (continuación)**
  - Un `.profile` típico

```
# -----  
#                               Variables de entorno  
# -----  
  
# Variables de entorno generales.  
  
OPENWINHOME=/usr/openwin  
export OPENWINHOME  
  
PATH=.:$OPENWINHOME/bin/xview:$OPENWINHOME/bin:/usr/ucb:/bin:/usr/bin:/usr/  
sbin:/usr/ccs/bin:/usr/lib/nis:/opt/frame/bin:/usr/local/bin  
export PATH  
  
MANPATH=$OPENWINHOME/man:/usr/man  
export MANPATH  
  
LD_LIBRARY_PATH=/usr/lib:$OPENWINHOME/lib:/usr/dt/lib  
export LD_LIBRARY_PATH  
  
# -----  
#                               Comandos a ejecutar por defecto.  
# -----  
  
# Definir máscara por defecto.  
  
umask 077  
  
# Tecla de borrado.  
  
stty erase '^H'  
  
# Definir un prompt (en bash).  
  
export PS1="\h \w"
```



## La shell (3)

- Un `.profile` típico (continuación)

```
# Inicialización de X Window.

# -----
#           Arrancar el servidor de X Window si es necesario.
# -----
#
# If possible, start the windows system
#
if [ `tty` = "/dev/console" ] ; then
    if [ "$TERM" = "sun" -o "$TERM" = "AT386" ] ; then

        if [ ${OPENWINHOME:-""} = "" ] ; then
            OPENWINHOME=/usr/openwin
            export OPENWINHOME
        fi

        echo ""
        echo "Starting OpenWindows in 5 seconds (type Control-C to interrupt)"
        sleep 5
        echo ""
        ${OPENWINHOME}/bin/openwin

        clear          # get rid of annoying cursor rectangle
        exit           # logout after leaving windows system

    fi
fi
```

- **Ficheros de inicialización/finalización (continuación)**

- Un `.login` típico.

- Idem pero sin las variables de entorno.
- Un ejemplo de prompt: `set prompt = "`uname -n`%~> "`
- También varía la parte de inicialización del servidor de X Window, y se suelen establecer alias (`alias l ls -al`).

- Un `.cshrc` típico.

- Suele dar valor a las variables de entorno (con `setenv` y sin `export`).
- El path también se puede especificar como `set path = ( dir1 dir2 ... dirN)`.
- También se podría haber hecho en el `.login`, pero tiene alguna ventaja hacerlo en `.cshrc`.

- Un `.logout` típico.

- Lo que interese antes de que se termine la sesión (ej.: `clear`).



## La shell (y 4)

- Ejecución de scripts dentro de la propia shell
  - sh, ksh, bash
    - `. nombreScript`
  - csh, tcsh
    - `source nombreScript`
- Utilidad típica
  - Cuando se hace un cambio a una variable de entorno de algún fichero de inicialización.
    - sh, ksh, bash => `. $HOME/.profile`
    - csh, tcsh => `source ~/.login, source ~/.cshrc`



# Caracteres especiales en Bourne shell (1)

- Concepto

```
mkdir tmp  
cd tmp
```

```
echo > f1  
echo > f2  
echo > f3
```

```
set -x
```

```
ls *  
+ ls f1 f2 f3
```

El shell expande (interpreta) el caracter \*,  
y posteriormente ejecuta el comando `ls` con  
los parámetros `f1 f2 f3`

```
f1 f2 f3
```

```
set +x
```

- Algunos caracteres especiales

Carácter	Significado
*	Reconoce cualquier cadena (que no empiece por <code>.</code> ), incluso la nula.
?	Reconoce cualquier carácter individual.
[ccc]	Reconoce cualquier carácter en <code>ccc</code> . Se admiten rangos (ej.: <code>[a-d0-3]</code> equivale a <code>[abcd0123]</code> ).
<code>\$var</code>	Valor de la variable de entorno <code>var</code> .
<code>\${var}</code>	Valor de la variable de entorno <code>var</code> ; evita confusión cuando está concatenada con texto.
"..."	Tomar <code>...</code> literalmente, interpretando <code>\$</code> , <code>\</code> y <code>`...`</code> .
'...'	Tomar literalmente <code>...</code> .
<code>`...`</code>	Ejecutar los comandos <code>...</code> y quedarse con la salida.



## Caracteres especiales en Bourne shell (2)

- Algunos caracteres especiales (continuación)

Carácter	Significado
\c	Tomar literalmente el caracter c. \<nuevaLinea> desecha el fin de línea.
#	Lo que sigue a # se considera comentario.
;	Ejecución secuencial de comandos. p1; p2 => primero p1, y después p2.
&	Ejecución en background. p1 & p2 & => p1 y p2 en paralelo con el shell.
( cmds )	Ejecuta comandos cmds en subshell.
{ cmds }	Ejecuta comandos cmds en la shell actual.

- Ejemplos

```
echo *
echo .*
echo ***
echo '***'
echo "***"
echo \*\*\*
```

```
ls ?ichero
ls [Ff]ichero
ls fichero[0-5A-B].txt
```

```
x=date
echo Esto es $x
echo 'Esto es $x'
echo "Esto es $x"
echo "Esto es ` $x ` "
echo Esto es ` $x `
ls -l `which ps`
```

```
echo $HOME--- mi directorio
echo ${HOME}--- mi directorio
```



## Caracteres especiales en Bourne shell (y 3)

- Ejemplos (continuación)

```
JDK_HOME=/opt/JDK
PATH=$PATH:$JDK_HOME/bin
CLASSPATH=$JDK_HOME/lib/classes.zip:.
```

```
export PATH          (si no estaba exportada ya)
export CLASSPATH    (si no estaba exportada ya)
```

```
ls; cat fichero1; rm fichero2
textedit & mailtool & pageview &
```



# Redireccionamientos de entrada/salida y pipes en Bourne shell (1)

- Todo proceso Unix dispone de:
  - Entrada estándar (0)
  - Salida estándar (1)
  - Error estándar (2)
- Un **filtro** es un programa que lee de la entrada estándar y escribe en la salida estándar (ej.: `cat`).
- La entrada estándar, la salida estándar y el error estándar se pueden redirigir.

- Ejemplos:

```
ps > ps.txt
cat < ps.txt > ps2.txt
cat > fichero
rm nombreDirectorio
rm nombreDirectorio > f
rm nombreDirectorio 2> f
```

- El shell interpreta las anteriores líneas, y redirige a los ficheros correspondientes.
- Un **pipe** permite conectar la salida estándar de un programa con la entrada estándar de otro.

```
ps -fea | grep pepe
```



## Redireccionamientos de entrada/salida y pipes en Bourne shell (2)

- Caracteres especiales de redirección y pipes.

Carácter	Significado
> fichero	Dirigir la salida estándar a <i>file</i> .
>> fichero	Concatenar la salida estándar con <i>fichero</i> .
< fichero	Tomar la entrada estándar de <i>fichero</i> .
p1   p2	Conectar la salida estándar de <i>p1</i> con la entrada estándar de <i>p2</i> .
n> fichero	Dirigir la salida del descriptor de fichero <i>n</i> al fichero <i>fichero</i> .
n>> fichero	Concatenar la salida del descriptor de fichero <i>n</i> al fichero <i>fichero</i> .
n>&m	Mezclar la salida del descriptor de fichero <i>n</i> con el descriptor de fichero <i>m</i> .
n<&m	Mezclar la entrada del descriptor de fichero <i>n</i> con el descriptor de fichero <i>m</i> .
<< s	Documento presente con sustitución. Leer de la entrada estándar hasta encontrar <i>s</i> al comienzo de una línea, interpretando \$, \ y `...`.
<< \s	Documento presente sin sustitución.
<< 's'	Documento presente sin sustitución.

- Ejemplos

```
echo Un ls -al de /home/fer > fichero
ls -al >> fichero
rm nombreDirectorio 2>> fichero
echo Error 1>&2
NombrePrograma > fichero 2>&1
```





## Redireccionamientos de entrada/salida y pipes en Bourne shell (y 3)

- Ejemplos (continuación)

```
grep buenas << Fin
> hola
> buenas
> adios
> Fin
```

```
VARIABLE=buenas
grep buenas << Fin
> hola
> $VARIABLE
> adios
> Fin
```

```
grep buenas << \Fin
> hola
> $VARIABLE
> adios
> Fin
```

```
sort fichero > fichero Cuidado !!
```

```
cat /etc/passwd | tr a-z A-Z | tee fichero | cut -f5 -d: | sort
```



# Expresiones regulares (1)

- **grep**

- Imprime las líneas que contienen un determinado patrón en ficheros de texto.
- `grep patron fichero1 ... ficheroN`
- Ejemplos:

```
grep palabra fichero
grep palabra *
grep -n variable *.[ch] (-n para imprimir números de línea)
ps -fea | grep -v fer (-v para buscar las líneas que no contienen el patrón)
grep -nv Fernando *
grep -i Fernando * (-i para no distinguir entre mayúsculas y minúsculas)

grep '^From:' /var/mail/fer
ls -l | grep '^d'
ls -l | grep '^.....rw'
grep '[Cc]asa' fichero
```

- Existen muchos comandos que entienden expresiones regulares (`grep`, `egrep`, `fgrep`, `sed`, `expr`, `awk`, etc.)
- Es recomendable poner todo el patrón entre comillas simples.
- Expresiones regulares básicas (`grep`, `egrep`, `sed`, `awk`, `expr`).

Expresión	Significado
<code>c o cad</code>	Un carácter o una cadena es una expresión regular.
<code>.</code>	Cualquier carácter.
<code>\c</code>	Cancela cualquier significado especial del carácter <code>c</code> .
<code>[lista]</code>	Reconoce cualquier carácter presente en la <code>lista</code> . Se admiten rangos (ej.: <code>[a-d0-3]</code> equivale a <code>[abcd0123]</code> ).
<code>[^lista]</code>	Cualquier carácter que no esté en la <code>lista</code> .



## Expresiones regulares (2)

- Expresiones regulares básicas (continuación)

ExprReg*	0 o más ocurrencias de ExprReg.
^ExprReg	Expresión regular al comienzo de línea.
ExprReg\$	Expresión regular al final de línea.

- Expresiones regulares limitadas (sed, grep).

- `\(ding\) \(\dong\) \1 \2 \1 \2` quiere decir `ding dong ding dong ding dong`.
- En general, la expresión se marca con `\(ExprReg\)` y se referencia con `\n`.

- Expresiones regulares completas (egrep, awk).

Expresión	Significado
ExprReg+	1 o más apariciones de ExprReg.
ExprReg?	0 o 1 aparición de ExprReg.
ExprRegA ExprRegB	ExprRegA o ExprRegB.
(ExprReg)	Agrupar la expresión regular ExprReg.

- Ejercicios:

- `ls -l | grep '^[^ ]* *[0-9]* *fer'`
- Buscar todas las líneas de un fichero que tengan los caracteres `[ o \`.
- Buscar todas las líneas de un fichero que terminen en `.`
- Contar el número de líneas de un programa C++ que no son comentarios simples.



## Expresiones regulares (3)

- **egrep**

- Parecido a `grep` (sin expresiones regulares limitadas, pero con expresiones completas).
- Admite opción `-f` para leer patrones de un fichero.
- Ejemplos:

```
egrep 'fer|felix' fichero
egrep '(fer)+' fichero
egrep -f erroresComunes.txt fichero.txt
```

- **fgrep**

- No admite metacaracteres.
- Utiliza un algoritmo de búsqueda muy rápido.
- También admite la opción `-f`.

- **sed**

- `sed 'lista de comandos' fichero1 ... ficheroN`
- Procesa líneas de ficheros de texto, y por cada línea aplica la lista de comandos.
- Escribe en la salida estándar.
- Principal uso: realizar sustituciones en ficheros de texto.
  - `sed 's/ExprReg/Sustituto/g' fichero1 > fichero2`
  - Error típico: `sed 's/ExprReg/Sustituto/g' fichero > fichero`
  - Ejemplos:

```
sed 's/Vien/Bien/g' f > g
sed 's/->/ /g f.cc > g.cc
sed 's/.$//' f > g
sed 's/doble/& &' f > g
sed 's/.*/(&)' f > g
```
- Otros usos:
  - `sed '/ExprReg1/s/ExprReg2/Sustituto/g' fichero1 > fichero2` (aplica el comando sólo a las líneas que contengan `ExprReg1`).
  - `sed '/ExprReg/q' fichero` (Imprime hasta encontrar `ExprReg`).
  - `sed '/ExprReg/d' fichero` (Imprime todas las líneas que no contengan `ExprReg`).



## Expresiones regulares (4)

- **sed** (continuación)

- Ejercicios:

- Asegurar que todas las ocurrencias de “casa” (ej.: “Casa”, “cAsa”, etc.) en un fichero se escriben como “casa”.
- Cambiar los comentarios C /\* .. \*/ (que abarquen sólo una línea) por los del tipo //.

- **find**

- Busca ficheros dentro de directorios.

- `find directorio1 ... directorioN expresión.`

- Una expresión está compuesta por operandos y operadores.

- Operandos:

- `-name fichero`
- `-perm num`
- `-type car`
- `-print`
- `-exec comando {} \;`

- Operadores:

- `!`, blanco, `-o`, `()`.

- Ejemplos:

```
find /home/fer -name EntornoUnix.fm -print
find . -perm 600 -print
find dir dir2 dir3 -name pepe.dat -perm 777 -print
find tmp2 \( -name '*.c' -o -name '*.o' \) -exec rm {} \;
```

- Ejercicio: intentar hacer un `grep` recursivo muy sencillo utilizando el comando `find` (utilizar opción `-type f`).



## Expresiones regulares (5)

- **expr**

- Evalúa argumentos.
- `expr argumento1 operador argumento2`
- Argumentos: números o cadenas.
  
- Operadores numéricos:
  - `+, -, *, /, %`
  - `variable='expr 3 \* 4'`
  - `echo $variable`
  - Utilidad típica: `contador='expr $contador + 1'`
  
- Operadores lógicos:
  - `>, >=, <=, <, <=, !=, &, |`
  - `expr 3 \< 4`
  - `echo $?` (0 si verdadero, 1 si falso, 2 si error de sintaxis).
  
- Búsqueda (empezando por la izquierda)
  - `expr cadena : ExprReg`
  - `echo $?` (0 si encontrada, 1 si no encontrada).
  - Imprime número de caracteres encontrados que concuerdan con la expresión regular.
  - `expr asa : '[aA]'` (imprime 1, `$? = 0`)
  - `expr asa12 : '[a-z]*'` (imprime 3, `$? = 0`)
  - `expr asa12 : asas` (imprime 0, `$? = 1`)
  - `expr 'El cielo es azul' : '.*'` (imprime la longitud de la cadena).
  - Cuidado con: `expr $variable : a`. Mejor con `expr "$variable" : a` o `expr X$variable : Xa`.



# Expresiones regulares (y 6)

- **awk**

- Es similar en concepto a sed, pero implementa todo un potente lenguaje de programación basado en C.

- `awk 'programa' fichero1 ... ficheroN`

- programa tiene el siguiente aspecto:

```
patrón { acción }  
patrón { acción }
```

- Para cada fichero de entrada, awk lee una línea a la vez. Para las líneas que contengan el patrón (expresión regular) se realiza la acción correspondiente.

- Ejemplo 1

```
ls -al | awk '  
  
/^d/ {print "Directorio: ", $0}  
/^-/ {print "Archivo normal: ", $0}  
  
' | more
```

- Ejemplo 2 (imprimir fichero de texto con las líneas en orden inverso)

```
cat file | awk '  
  
{ line[NR] = $0 }  
  
END {  
    for (i=NR; i>0; i--)  
        print line[i]  
  
'
```



# Programación de scripts en Bourne Shell (1)

- Un script es un fichero de texto que contiene un conjunto de sentencias de la shell (`if`, `case`, `for`, `while`, etc.) y/o comandos.
- Se utilizan para implementar comandos de forma “rápida”.
- Deben llevar permiso de ejecución.
- Ejemplo:

```
#!/bin/sh
#
# Hace los mismo que "ls".

for i in *
do
    echo $i
done
```

- **Parámetros**

- Al igual que un programa cualquiera, un script puede recibir parámetros.
- Ej.: `miScript par1 par2 .. parN`

Notación	Significado
<code>\$0</code>	Nombre del script
<code>\$1, \$2, ..., \$9</code>	Primer parámetro, segundo parámetro, ..., noveno parámetro.
<code>\$#</code>	Número de parámetros.
<code>shift [n]</code>	Desplaza parámetros a la izquierda (y disminuye <code>\$#</code> en <code>n</code> unidades).
<code>set p1 .. pn</code>	Establece <code>set p1 .. pn</code> como parámetros.





## Programación de scripts en Bourne Shell (2)

- Parámetros (continuación)

- Ejemplo 1

```
echo $0 $1 $2
set x y z
echo $0 $1 $2
```

- Ejemplo 2

```
contador=0
while test $# -gt 0
do
    contador=`expr $contador + 1`
    echo Parametro $contador: $1
    shift
done
```

- Ejemplo 3

```
contador=0
for parametro in "$@"
do
    contador=`expr $contador + 1`
    echo Parametro $contador: $parametro
done
```

- Algunos parámetros más ...

Notación	Significado
"\$@"	Todos los parámetros.
\$\$	PID (Process Identifier) del proceso.
\$?	Código de retorno del último comando.



## Programación de scripts en Bourne Shell (3)

- Construcción **if**

- Sintaxis

```
if condición; then acciones; fi
if condición; then acciones; else acciones2; fi
if condición; then acciones; elif condición2; then acciones2; ... ; fi
```

- condición es una lista de comandos (separados por “;” o <fin de línea>), donde el código de retorno (\$?) del último comando indica el valor de la condición: verdadero (0) o falso (cualquier valor distinto de 0). **Es la convención contraria a otros lenguajes (ej.: C/C++).**

- Ejemplo 1

```
numFicheros=0
numDirectorios=0

for i in "$@"
do
    if test -f $i; then
        numFicheros=`expr $numFicheros + 1`
        echo \"$i\" es un fichero
    elif test -d $i; then
        numDirectorios=`expr $numDirectorios + 1`
        echo \"$i\" es un directorio
    else
        echo No se lo que es \"$i\"
    fi
done

echo Número de ficheros: $numFicheros
echo Número de directorios: $numDirectorios
```

- Ejemplo 2

```
if grep -l $1 $2 > /dev/null; then
    echo Se encontró $1 en $2
fi
```



# Programación de scripts en Bourne Shell (4)

- Comando **test**

- Se utiliza típicamente en las sentencias que contienen alguna condición (`if`, `while`, `until`).
- Sintaxis: `test expresión` (ej.: `test -f fichero`) o `[ expresión ]` (ej.: `[ -f fichero ]`).
  - Ficheros
    - `-r fichero`, `-w fichero`, `-x fichero`, `-f fichero`, `-d directorio`.
  - Cadenas
    - `-z cadena`, `-n cadena`, `cadena1 = cadena2`, también `!=`, `<`, `>`.
  - Enteros
    - `entero1 -eq entero2`, también `-ne`, `-lt`, `-le`, `-ge`, `-gt`.
  - Expresiones
    - `!expresión`, `expresión1 -a expresión2`, `expresión1 -o expresión2`.
  - Las condiciones se pueden agrupar con `( grupo )`.

- Construcción **case**

- Sintaxis

```
case palabra in
    patrón [|patrón] ...) acciones;;
    patrón [|patrón] ...) acciones;;
    ...
esac
```

- Las reglas de construcción de patrones son similares a las de reconocimiento de nombres de ficheros.
- Ejemplo 1

```
variable=B
case $variable in
    A) echo Es una "A";;
    B|C) echo Es una "B" o una "C";;
    *) echo Es otra letra;;
esac
```



## Programación de scripts en Bourne Shell (5)

- Construcción **case** (continuación)
  - Ejemplo 2: Una versión mejorada de `cal`.

```
#!/bin/sh
# ical (improved cal): Mejor interfaz para el comando "cal"
# Uso: ical [ mes [ anho ] ]

# Comprobar qué hay que hacer en función del número de
# parámetros.

case $# in

0) set `date`; mes=$2; anho=$6;;
1) mes=$1; set `date`; anho=$6;;
2) mes=$1; anho=$2;;
*) echo "ical: Uso: ical [ mes [ anho ] ]" 1>&2
   exit 1;;

esac

# Obtener el mes en formato numérico (si no lo estaba).

case $mes in

[Jj]an*)      mes=1;;
[Ff]eb*)      mes=2;;
[Mm]ar*)      mes=3;;
[Aa]pr*)      mes=4;;
[Mm]ay*)      mes=5;;
[Jj]un*)      mes=6;;
[Jj]ul*)      mes=7;;
[Aa]ug*)      mes=8;;
[Ss]ep*)      mes=9;;
[Oo]ct*)      mes=10;;
[Nn]ov*)      mes=11;;
[Dd]ec*)      mes=12;;
[1-9]|10|11|12) ;;
*)            echo "ical: Especificación incorrecta de mes" \
              1>&2
              exit 1;;

esac
```



## Programación de scripts en Bourne Shell (6)

- Ejemplo 2: Una versión mejorada de `cal` (continuación).

```
# Invocar a "cal" para obtener el calendario del mes
# solicitado.

if cal $mes $anho 2>/dev/null; then
    exit 0
else
    echo "ical: Especificación incorrecta de anho" 1>&2
    exit 1
fi
```

- Construcción **for**.

- Sintaxis

```
for variable [in lista de palabras]; do acciones; done
```

- Ejemplo 1

```
for i
do
    echo $i
done
```

es lo mismo que ...

```
for i in "$@"
do
    echo $i
done
```



# Programación de scripts en Bourne Shell (7)

- Construcción **for** (continuación)

- Ejemplo 2

```
for i in palabra1 palabra2 palabra3; do
    echo $i
done
```

- Ejemplo 3

```
for i in *.cc *.h; do
    echo $i
done
```

- Ejemplo 4

```
#!/bin/sh
# pick argumento ...
# pick: selección de argumentos.

for i
do
    printf "$i ? " > /dev/tty
    read respuesta

    case $respuesta in
    y*) echo $i;;
    q*) break;;
    esac
done
```

Ahora es posible hacer cosas como: `rm `pick *.c``, `lpr `pick *.c``, etc.



## Programación de scripts en Bourne Shell (8)

- Construcción **while**.

- Sintaxis

```
while condición; do acciones; done
```

- condición se interpreta de la misma forma que en la construcción **if**.

- Ejemplo 1

```
while sleep 60
do
    who | grep fer
done
```

- Ejemplo 2

```
while read linea; do
    echo $linea
done < fichero
```

- Construcción **until**.

- Sintaxis

```
until condición; do acciones; done
```

- condición se interpreta de la misma forma que en la construcción **if**.

- Ejemplo

```
until who | grep fer
do
    sleep 60
done
```



## Programación de scripts en Bourne Shell (9)

- Funciones

- Idea similar a las funciones en un lenguaje procedural.
- Se accede a los parámetros de la función utilizando la misma notación ("\${@}", "\${#}", \$0, \$1, etc., shift) y comandos utilizados para acceder a los parámetros del script.
- Las funciones pueden devolver un valor de resultado (return entero).
- Ejemplo

```
Funcion ()
{
    echo "Número argumentos: $#"
```

```
    for i in "${@}"; do
        if [ $i = adiós ]; then
            return 0
        fi
    done
    return 1
}
```

```
if Funcion unos dos adiós tres cuatro; then
    echo Uno de los parámetros era \"adiós\"
fi
```





## Programación de scripts en Bourne Shell (10)

- Algunas cosas más ...
  - `for`, `do`, `while` se pueden redireccionar (ejemplos anteriores).
  - `break`. Abandona un bucle (`for`, `do`, `while`).
  - `continue`. Salta a la condición de un bucle (`for`, `do`, `while`).
  - `return`. Se puede utilizar sin argumentos para abandonar la ejecución de una función.
  - La entrada, salida y error estándar de las funciones se pueden redirigir.

```
Funcion ()  
{  
    echo "Hola"  
}
```

```
Funcion > hola.txt
```

- `unset variable`. Elimina variable del entorno.
- `c1 && c2` es equivalente a:

```
if c1; then  
    c2  
fi
```

- `c1 || c2` es equivalente a:

```
c1  
if [ $? -ne 0 ]; then  
    c2  
fi
```



## Programación de scripts en Bourne Shell (11)

- `c1 || c2` es equivalente a (continuación)

- Ejemplo

```
mkdir directorio 2> /dev/null || {ImprimirMensajeError; exit 2}
```

es lo mismo que ...

```
mkdir directorio 2> /dev/null
if [ $? -ne 0 ]; then
    ImprimirMensajeError
    exit 2
fi
```

- Comandos `true` ( `$? es 0`) y `false` ( `$? es distinto de cero`).

- Ejemplo

```
while true; do
    <<comandos>>
done
```

- Ejercicio: `tab2blank1`.

- `tab2blank1 fichero [fichero ...]`
- Sustituye cada tabulador por cuatro blancos.
- Si uno de los parámetros es un directorio, se lo salta.
- Controla errores (argumentos incorrectos y acceso a ficheros).



## Programación de scripts en Bourne Shell (12)

- Ejercicio: tab2blank1 (continuación).

```
#!/bin/sh
# -----
# tab2blank1 fichero [fichero ...]
#
# FUNCION: Sustituye tabuladores por 4 blancos.
# VALOR DE RETORNO:
#   * 0 => OK.
#   * >0 => Hubo algún problema.
# -----

# -----
# Definición de constantes.
# -----

NOMBRE_PROGRAMA=tab2blank1
BLANCOS="      "

# Errores

CR_OK=0
CR_USO=1
CR_FICHERO=2

# -----
# Funciones.
# -----
```



## Programación de scripts en Bourne Shell (13)

- Ejercicio: tab2blank1 (continuación).

```
ImprimirMensajeError ()

# -----
# FUNCION: Imprime el mensaje de error $1 en el estándar error.
# -----

{
    case $1 in
        $CR_USO)
            echo "Uso: $NOMBRE_PROGRAMA fichero [fichero ...]" 1>&2;;
        $CR_FICHERO)
            echo "No se puede leer o escribir en $2" 1>&2;;
    esac
}

VerificarSintaxis ()

# -----
# FUNCION: Comprueba que la sintaxis de llamada es correcta.
# -----

{
    if [ $# -eq 0 ]; then
        ImprimirMensajeError $CR_USO
        exit $CR_USO
    fi
}
```



## Programación de scripts en Bourne Shell (y 14)

- Ejercicio: tab2blank1 (continuación).

```
# -----  
# Programa principal.  
# -----  
  
# Verificar sintaxis de invocación.  
  
VerificarSintaxis "$@"  
  
# Inicializar variables.  
  
error=$CR_OK  
  
# Procesar ficheros.  
  
for i  
do  
    if [ ! -d $1 ]; then  
        if [ -r $i -a -w $i ]; then  
            sed "s//$BLANCOS/g" $i > /tmp/$NOMBRE_PROGRAMA.$$  
            cp /tmp/$NOMBRE_PROGRAMA.$$ $i  
            rm /tmp/$NOMBRE_PROGRAMA.$$  
        else  
            ImprimirMensajeError $CR_FICHERO $i  
            error=$CR_FICHERO  
        fi  
    fi  
done  
  
exit $error
```



# Acceso remoto (1)

- **ftp** (File Transfer Protocol)
  - Permite transferir ficheros entre máquinas remotas.
  - La máquina remota tiene que disponer de servidor de FTP.
  - `ftp orca.gris.des.fi.udc.es`
  - Comandos
    - `ls, mls, dir, mdir, cd, lcd, pwd, mkdir, rmdir.`
    - `!<comando local>`.
    - `binary (bin)` (para transferencias binarias), `ascii` (para transferencias en ASCII).
    - `get, mget.`
    - `put, mput.`
    - `prompt`
    - `help`
    - `open, close.`
    - `bye, quit.`
    - `delete, mdelete, rename.`
- Un script para hacer un ftp a una hora dada ...

```
$ cat fichero
```

```
ftp << FIN
```

```
open olmo.master.abrente.es
cd /Trabajos/tmp
prompt
mget *
quit
```

```
FIN
```

```
$ cat $HOME/.netrc
```

```
machine olmo.master.abrente.es login anonymous password fer@gris.des.fi.udc.es
machine ftp.omg.org login anonymous password fer@gris.des.fi.udc.es
```

```
$ at -s -f fichero 00:00 Oct 26
```

```
909442800.a      Tue Oct 26 00:00:00 1998
```

at envía un mail con el resultado de la operación (la opción `-s` es para que ejecute el script con el Bourne Shell).



## Acceso remoto (2)

- Un script para hacer un ftp a una hora dada ... (continuación)

```
$ at -l
909442800.a      Tue Oct 26 00:00:00 1998
$ at -r 909442800.a
```

- Existen sitios que disponen de “ftps anónimos”.
  - Usuario: “anonymous” y password: <dirección correo electrónico>.
  - Proporcionan acceso a información pública.

- **telnet**

- Conexión a una máquina remota (que disponga de servidor apropiado).
- telnet orca.gris.des.fi.udc.es

- **rsh**

- Ejecutar un comando en otra máquina (que disponga de servidor apropiado).
- rsh [-l login] maquina comando
- Para que no pida la password (útil en scripts), se necesita fichero \$HOME/.rhosts en la máquina destino, concediendo acceso

```
$ rsh -l fbellas nogal.master.abrente.es ls Trabajos
(desde fer@quenlla.gris.des.fi.udc.es)
```

```
$ cat $HOME/.rhosts (en fbellas@nogal.master.abrente.es)
quenlla.gris.des.fi.udc.es fer
```

```
$ cat $HOME/.rhosts (en fbellas@nogal.master.abrente.es)
+ fer
```



## Acceso remoto (3)

- **rcp**

- Copia remota de ficheros (requiere servidor apropiado en máquina remota).
- Se ejecuta mediante `rsh`.

```
rcp tab2blank2 fer@quenlla.gris.des.fi.udc.es
rcp tab2blank2 fer@quenlla.gris.des.fi.udc.es:bin
rcp tab2blank2 fer@quenlla.gris.des.fi.udc.es:/tmp
rcp -r Java fer@quenlla.gris.des.fi.udc.es:Trabajos
rcp -r fer@quenlla.gris.des.fi.udc.es:Trabajos/Java .
```

- **rlogin**

- Entrar en una máquina remota (que disponga de servidor apropiado).
- `rlogin [-l login] maquina`
- Mismas consideraciones que `rsh` con respecto a `$HOME/.rhosts`.

- **finger**

- Visualizar información acerca de usuarios locales y remotos (requiere servidor apropiado en máquina remota).
- `finger fer`
- `finger fer@quenlla.gris.des.fi.udc.es`

- **mailx y herramientas gráficas**

- Lectores de correo electrónico.

- **talk**

- Conversación con otro usuario (require servidor apropiado en máquina remota).
- `talk fer@quenlla.gris.des.fi.udc`





## Acceso remoto (4)

- **Protocolos SSH (Secure SHell)**

- `ftp`, `rcp`, `telnet`, `rlogin` y `rsh` no usan ningún tipo de cifrado (envían toda la información en claro, inclusive las passwords).
- El uso de ficheros `$HOME/.rhosts` tampoco es una solución segura.
- Estas limitaciones, especialmente el envío de passwords no cifradas y el uso de ficheros `$HOME/.rhosts`, no son aceptables cuando cuando se proporciona acceso remoto desde Internet.
- SSH es un conjunto de protocolos (**distintos de los anteriores**) que permiten la misma funcionalidad transmitiendo la información cifrada.
- Una de las implementaciones más famosas de los protocolos SSH es OpenSSH, que proporciona los comandos clientes `sftp`, `scp` y `ssh`, el servidor y herramientas de gestión de claves.
  - <http://www.openssh.com>
  - <http://www.openssl.org>
  - <http://www.freessh.org> (software para varias plataformas)

- **Ejemplos**

```
sftp fbellas@nogal.master.abrente.es
sftp -l fbellas nogal.master.abrente.es
scp tab2blank2 fbellas@nogal.master.abrente.es:bin
scp -r fbellas@nogal.master.abrente.es:Trabajos/Java .
ssh fbellas@nogal.master.abrente.es
```

- **Para que ninguno de los comandos pida password (útil en scripts), se pueden generar pares clave pública/privada. Ejemplo:**

- \* Cuenta desde la que se accede: `fer@quenlla.gris.des.fi.udc.es`
- \* Cuenta a la que se desea acceder: `fbellas@nogal.master.abrente.es`
- \* En `fer@quenlla.gris.des.fi.udc.es =>` ejecutar `"ssh-keygen -t dsa"`, introduciendo un passphrase de entre 10-30 caracteres.
- \* Idem en `fbellas@nogal.master.abrente.es`
- \* Añadir el contenido de `$HOME/.ssh/id_dsa.pub` (en `fer@quenlla.gris.des.fi.udc.es`) a `$HOME/.ssh/authorized_keys2` (en `fbellas@nogal.master.abrente.es`), creando este último si es que no existía.
- \* En `fer@quenlla.gris.des.fi.udc.es =>` ejecutar `exec ssh-agent $SHELL`  
`ssh-add` (introducir passphrase)
- \* Desde **este shell** cada vez que se ejecute `sftp`, `scp` o `ssh` sobre `fbellas@nogal.master.abrente.es` no se pedirá password



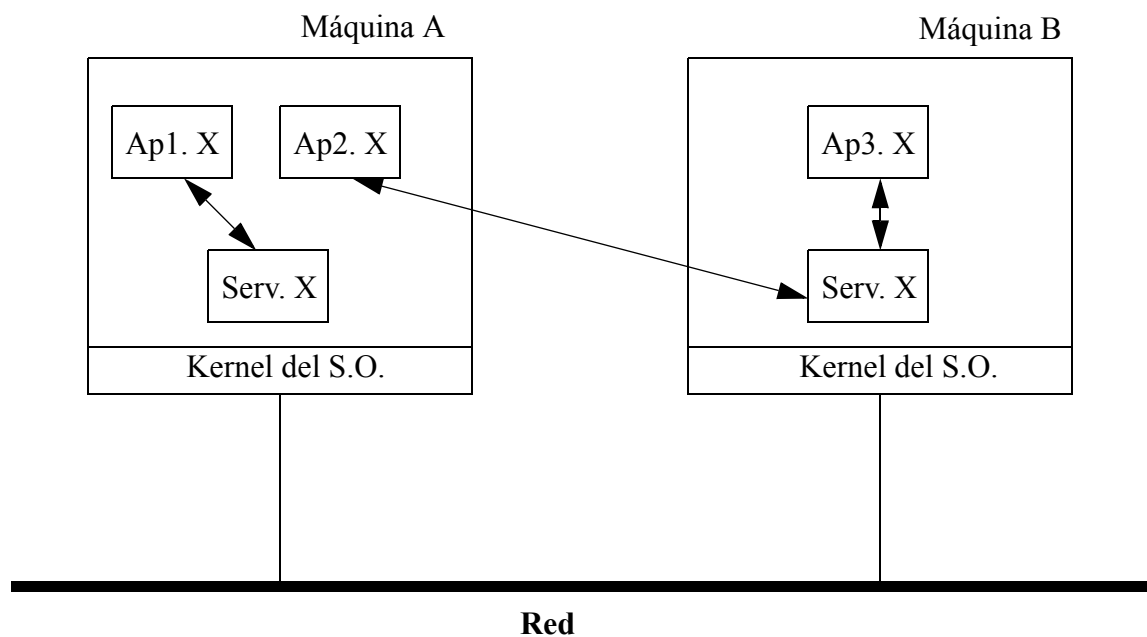
## Acceso remoto (y 5)

- **Protocolos SSH (cont)**
  - Si no se hubiese ejecutado `ssh-agent` y `ssh-add`, los comandos `sftp`, `scp` y `ssh` pedirían la passphrase.
  - `ssh-keygen` permite generar un par clave pública (`$HOME/.ssh/id_dsa.pub`) / privada (`$HOME/.ssh/id_dsa`).
  - `ssh-agent` permite gestionar un conjunto de identidades (claves privadas y passphrases); puede ejecutar un programa (típicamente un shell), que heredará una variable de entorno que otras aplicaciones (ej.: `ssh`, `scp`, `sftp`) pueden usar para comunicarse con él.
  - `ssh-add` permite añadir una identidad (por defecto `$HOME/$HOME/.ssh/id_dsa`) y una passphrase a `ssh-agent`.
  - Es posible integrar `ssh-agent` y `ssh-add` en entornos gráficos (ej.: Gnome), para que no tener que ejecutarlos desde cada terminal.



# Arquitectura X Window (1)

- Sistema de ventanas basado en el modelo cliente/servidor.
- Disponible no sólo para Unix, sino también para otros sistemas operativos.
- Cada máquina ejecuta un servidor de X Window.
- Las aplicaciones X Window se comunican con el servidor para dibujar en la pantalla (crear ventana, dibujar línea, etc.).
- Bajo este entorno, es posible visualizar una aplicación X Window en una máquina distinta a la máquina que ejecuta la aplicación.



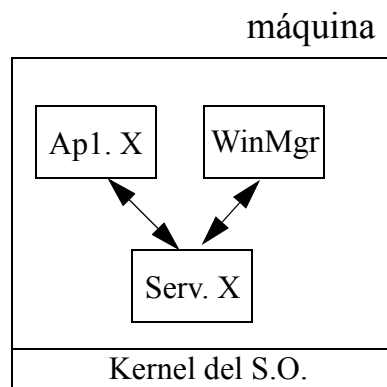
## Arquitectura X Window (2)

- Un aplicación X se comunica con el servidor X al que hace referencia la variable de entorno DISPLAY.

```
setenv DISPLAY quenlla:0.0
```

```
DISPLAY=quenlla:0.0  
export DISPLAY
```

- **Control de acceso: xhost.**
    - El usuario que está en la consola de una máquina puede controlar qué otras máquinas pueden acceder a su servidor de X Window (es decir, visualizar aplicaciones en su pantalla).
  - xhost -
  - xhost +
  - xhost - maquina
  - xhost + maquina
  - xhost
- **Window manager**
    - Aplicación X especial.
    - El servidor X le informa de eventos relativos a otras aplicaciones.
    - Añade “adornos” al resto de aplicaciones X (entre otras cosas ...).



## Arquitectura X Window (y 3)

- Distintas librerías de acceso a X Window, distintos window managers => Motif, OpenWindows, CDE, etc.
- Existen servidores X Window para Microsoft Windows, que permiten visualizar aplicaciones remotas X en PC.



# Arquitectura típica de una red heterogénea Unix/MS-Windows (1)

- Conceptos (en una red Unix)
  - NFS (Network File System)
    - Permite compartir sistemas de ficheros/directorios entre distintas máquinas.
    - Los demonios de NFS se ejecutan en cada máquina.
  - Servidor de correo
    - Gestiona los correos que llegan a un dominio (`gris.des.fi.udc.es`).
    - Se ejecuta en una máquina de la red.
    - `/var/mail` compartido (NFS) por todas las máquinas Unix.
  - Servidor de DNS (Domain Name Server)
    - Establece un mapping entre direcciones IP (`193.144.50.190`) y nombres lógicos (`orca`) para un dominio dado (`gris.des.fi.udc.es`).
    - Se ejecuta en una máquina de la red.
  - Servidor de NIS+
    - Gestiona información de manera centralizada (ej.: las passwords).
    - Se ejecuta en una máquina de la red.
  - Servidor samba
    - Permite compartir recursos Unix (directorios, impresoras, passwords, etc.) con máquinas con sistema operativo Microsoft Windows.
    - Se ejecuta en una máquina de la red.
  - Servidor POP3
    - Permite que aplicaciones que entienden el protocolo POP3 puedan acceder al correo electrónico (ej.: Netscape Communicator).
    - Basta con tener una máquina en la red con un servidor POP3.



# Arquitectura típica de una red heterogénea Unix/MS-Windows (y 2)

- En la red del máster ...
  - manzano (PC con Solaris x86)
    - Servidor de correo
    - Servidor de DNS
    - Servidor de NIS+
    - Servidor de POP3
  - nogal (Sun Ultra-2 con Solaris SPARC)
    - Exporta los directorios /export/home (contiene las cuentas de usuario).
    - Exporta /var/mail, /usr/local/sparc, /usr/local/i386 y /opt.
  - olmo (PC con Windows NT)
    - Gestiona cuentas con NT.
  - Máquinas del laboratorio
    - Disponen de Windows 98 y Solaris x86.
    - Es posible acceder al directorio /home/nombreDeUsuario desde Windows 98.
    - Solaris x86 monta los directorios que exporta nogal (el directorio de cada usuario se monta como /home/nombreUsuario).
- Distintos tipos de configuración de Solaris
  - Standalone.
  - Server.
  - Dataless.
  - AutoClient.
  - Diskless.



# Recursos

- **Bibliografía**
  - Brian W. Kernighan, Rob Pike, *El Entorno de Programación Unix*, Prentice Hall, 1987.
  - Libros específicos ...
- **Mi página web.**
  - <http://www.tic.udc.es/~fbellas>
  - Transparencias, ejemplos, enunciado de la práctica, etc.

