# APUNTES SISTEMA OPERATIVO UNIX

## Índice General



# Introducción

## Generalidades

Un sistema operativo es un programa que se encarga de administrar los recursos que nos proporciona el computador, por ejemplo: discos, impresora, terminal, etc., los cuales son conocidos también como **dispositivos**.

Unix es lo que se llama un sistema operativo **multi-usuario** y **multi-tarea**, esto quiere decir que en un mismo computador trabajan simultáneamente varias personas (**multi-usuario**) y, además, cada uno de ellas puede ejecutar varios comandos a la vez (**multi-tarea**).

Cada usuario tiene asociada una cuenta en la cual puede trabajar independiente de otros, prácticamente

el usuario no se da cuenta que el computador está siendo compartido. Además, el usuario dispone de un espacio en disco, en el cual puede guardar la información que desee como si fuera un diskette de microcomputadores.

El principal objetivo de este apunte es presentar una visión general, de las características que nos ofrece Unix más que de sus comandos, los que en algunos casos, no vienen con una explicación muy completa. En el capítulo 8 se explica como poder obtener mayor información sobre los comandos y el Apéndice A presenta una lista de los más utilizados con una pequeña descripción.

### Historia

Unix fue desarrollado originalmente por Ken Thompson, en los laboratorios Bell, entre 1968 y 1970 escribió la primera versión de Unix para un computador PDP-7 de Digital Equipment Corp., tiempo después se le unió Dennis Richie, quien trabajaba en el diseño de un nuevo lenguaje, el **"C"**, juntos escribieron Unix para el computador PDP-11, luego lo reescribieron en lenguaje **"C"** (1973), así fácilmente se podía trasladar a otros computadores.

Alrededor de 1974, otras organizaciones, como la Universidad de Berkeley, obtuvieron licencia sobre Unix. En 1978 aparece la primera versión comercial de Unix, conocida como Unix versión 7 (Unix/V7). Con el paso del tiempo, Unix se perfecciona, con lo que en 1982 aparece Unix System III, y en 1983 Unix System V. En la actualidad, las versiones de Unix más utilizadas cumplen con el standard Unix BSD, de la Universidad de Berkeley, tales como Ultrix, SunOs y otras.

## Conceptos previos

### Archivos

Un archivo es sólo una secuencia de caracteres (letras, dígitos, símbolos especiales, etc.), cuyo largo se mide en **Bytes** (cada caracter se representa en un **Byte**), en ellos podemos guardar la información que nosotros necesitemos. La identificación de cada uno de ellos se hace mediante un nombre, que puede ser una secuencia de letras, números y caracteres especiales de cualquier largo, aunque las versiones antiguas de Unix lo limitan a 14 caracteres, la idea es que el nombre que le demos al archivo represente de forma breve su contenido. Por ejemplo, algunos nombres de archivos podrían ser:

```
apunte.unix
carta.Erika
saludo
Notas
```

### Directorios

El objetivo de los directorios es el de organizar nuestros archivos, podríamos considerar el disco de nuestro computador como un gran cajón, en que tenemos varias carpetas, dentro de las cuales pueden haber otras carpetas y/o fichas, así, estas carpetas representan los directorios y las fichas a los archivos, es decir, podemos crear nuevos directorios y dentro de ellos colocar otros directorios y archivos en número que generalmente sobrepasa nuestras necesidades. Con esto se obtiene una estructura similar a la de un **Árbol genealógico**, con la salvedad que cada directorio tiene un solo **padre**. El directorio de donde parten todas las ramas es conocido como **Raíz** (Root).

Las reglas a seguir para identificar un directorio son las mismas que para un archivo, también se recomienda que el nombre del directorio de alguna idea de los archivos que contiene, por ejemplo, podríamos tener el directorio presupuestos y dejar dentro de él todos los archivos que tengan relación con presupuestos.

Cada usuario cuenta con un directorio, dentro de él puede grabar sus archivos y crearse más directorios para poder clasificar su información.

Unix utiliza varios directorios para su funcionamiento, de los cuales podemos mencionar:

**/bin:** En él se guardan los comandos de Unix (como programas ejecutables).

**/dev:** Con los archivos contenidos en este directorio se simulan los dispositivos externos, como son el terminal, la impresora, etc.

**/usr:** Contiene los directorios de los diferentes usuarios.

**/etc:** Contiene archivos útiles para el sistema pero que no están sujetos a una clasificación.

**/tmp:** Contiene archivos que se utilizan en forma temporal (se borrarán dentro de poco).

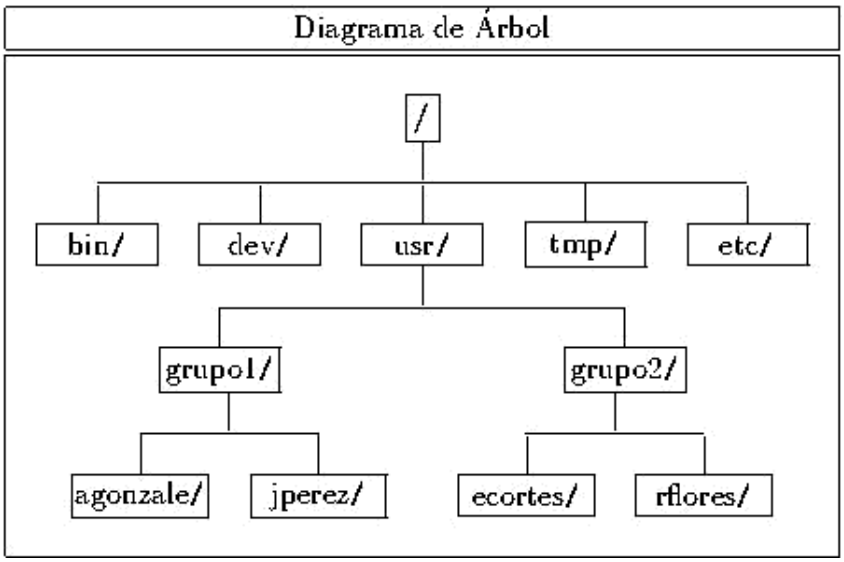


Figura 1

La identificación de un archivo en especial se puede hacer por medio del **camino** (*PATH*) que es necesario recorrer para alcanzar dicho archivo, en el camino se especifican los directorios que hay que recorrer separando cada uno de ellos con el caracter / (*slash*). Así, según el diagrama dado en la figura 1, para alcanzar el archivo carta.Erika, podría darse de la forma:

```
/usr/grupo1/jperez/carta.Erika
```

Pero esta forma, llamada forma absoluta ya que no depende del directorio en que estemos trabajando, es

bastante incómoda, sobre todo si el camino que hay que recorrer es largo, como se acostumbra en Unix, por ello se prefiere la forma relativa, la cual depende del directorio en el cual estemos trabajando. Al conectarse un usuario en una cuenta queda trabajando en el directorio que se le asignó para ello, el cual tiene el mismo nombre que la cuenta y está ubicado en algún directorio que depende de la distribuición que haya decidido el administrador. Para el caso del ejemplo, nuestro usuario **jperez** tiene asignado el directorio: **/usr/grupo1/jperez**; al momento de conectarse en la cuenta ya está trabajando en ese directorio, por lo que alcanzar el archivo es mucho más sencillo, pues basta con sólo su nombre: **carta.Erika** ya que se encuentra dentro del directorio actual. Similarmente, si estuvieramos trabajando en el directorio **/usr/grupo1**, para referirnos al mismo archivo se haría de la forma:

```
jperez/carta.Erika
```

El caracter / al comienzo indica que el nombre del archivo será dado a partir del directorio raíz (forma absoluta), de lo contrario asume que el camino es tomado a partir del directorio de trabajo.

Debemos notar que cada directorio (salvo la raíz) tiene definidos dos directorios con un significado especial (por lo tanto no se pueden borrar), ellos son:

- **.** : Se refiere al directorio que estamos referenciando.
- **..** : Se refiere al directorio padre, es decir, donde éste fue creado.

## Caracteres de Globalización

Los caracteres de globalización son caracteres que Unix utiliza para que el usuario pueda señalar un grupo de archivos que cumplan con un cierto patrón, dicho patrón es reemplazado automáticamente por los nombres de archivos existentes que lo cumplan antes de llamar al comando. Los caracteres de globalización (Wilcards o Metacaracteres) más simples son:

- * : Se reemplaza por cualquier secuencia de caracteres.
- ? : Se reemplaza por una caracter cualquiera.

Si por ejemplo tenemos los archivos:

caracol
carta
carta1
carta2
carta3
carta7
hola
marta

Y suponiendo que tenemos un comando llamado **procesar** que realiza algún trabajo sobre los archivos que se le dan como parámetros podríamos dar los siguientes comandos con los siguientes alcances:

**procesar*** : procesa todos los archivos del directorio actual.

**procesar car*** : procesa los archivos *caracol, carta, carta1, carta2, carta3* y *carta7*.

**procesar ?arta** : procesa los archivos *carta* y *marta*.

**procesar ?ar*** : procesa los archivos *caracol, carta, carta1, carta2, carta3 carta7* y *marta*.

**procesar carta?** : procesa los archivos *carta1, carta2, carta3* y *carta7*.

**procesar carta*** : procesa los archivos *carta, carta1, carta2, carta3* y *carta7*

También se pueden utilizar los paréntesis cuadrados ( [ ] ), indicando que en esa posición puede ir algún caracter de los que están dentro de ellos, así por ejemplo:

**procesar carta[1237]** : procesa los archivos *carta1, carta2, carta3* y *carta7*.

**procesar ?[aeiou]*** : procesa todos los archivos cuya segunda letra sea una vocal minuscula.

Cuando se utilizan los paréntesis cuadrados y dentro de ellos se tienen rangos de caracteres consecutivos, se pueden señalar de la forma [primero-último], así se tiene que:

| | | |
|---|---|---|
| **[0-9]** | s equivalente a | [0123456789] |
| **[a-z]** | es equivalente a | [abcdefghijklmnopqrstuvwxyz] |
| **[A-Z]** | es equivalente a | [ABCDEFGHIJKLMNOPQRSTUVWXYZ] |
| **[a-gK-O4-7]** | es equivalente a | [abcdefKLMNO4567] |

# Conexión y Desconexión:

Lo primero que se debe hacer para comenzar a trabajar es conectarse a la cuenta definida para el usuario, por lo general se dispone de una cuenta por persona. Al acercarnos a un terminal debe verse un mensaje del tipo:

```
login:
```

Esto nos indica que el computador está en espera de una conexión, en este momento podemos escribir nuestro **username** (nombre de usuario), por ejemplo:

```
login: jperez
```

Luego si pulsamos la tecla **ENTER** nos responde con el mensaje:

```
password:
```

Lo que nos está pidiendo es una palabra clave (similar al sistema utilizado por los cajeros automáticos), cada usuario puede definirse su propia password que debe ser conocida sólo por él, así impide que otras personas ocupen su cuenta, lean, modifiquen o borren sus archivos o lean su correo personal. En este momento debemos ingresar nuestra password (la primera vez es asignada por el administrador del computador), la cual no se muestra en la pantalla mientras se escribe, así se evita que otras personas la

puedan leer. En caso que algo falle, el computador nos muestra el mensaje:

```
login incorrect
```

Y vuelve a preguntar por un username, esto puede significar que se ha cometido un error al digitar el nombre de usuario o la password por lo que volvemos a intentar. Si luego de varios intentos aún no logramos conectarnos es recomendable ir a hablar con el administrador.

**Debemos notar que Unix es** *case sensitive***, es decir, considera las letras mayúsculas y minúsculas distintas, por ejemplo, considera las palabras hola, HOLA y HoLa distintas.** Por ello debemos poner especial atención al ingresar nuestra identificación, pues es causa común de errores.

En caso de haber ingresado los datos correctamente, aparecerán en la pantalla algunos mensajes del administrador (si existen) y luego el **prompt** del sistema (mensaje que nos indica que el computador está esperando un comando), algo como:

- %
- [araucaria:~/] %

La primera forma, en la práctica, ya no es muy utilizada, generalmente uno quiere más información en el prompt, como por ejemplo: el computador en que estamos trabajando, la cuenta en que estamos, el directorio que estamos utilizando, etc.

Cada vez que se mencione algún ejemplo en este manual irá precedido por un prompt %, con el fin de indicar que el comando debe ser escrito en la línea de comandos.

En este momento se está ejecutando en nuestra cuenta el **Shell** que es un programa que interpreta los comandos y los ejecuta, él se encarga de la comunicación entre el usuario y el computador, de otra forma, sería casi imposible darle órdenes. Por lo general, se dispone en Unix de varios Shell, como por ejemplo, el **Bourne Shell** (sh) que es el más básico y proporciona menos facilidades para el usuario, y el **C-Shell** (csh).

La forma de indicarle al computador lo que queremos hacer es por medio de órdenes o comandos que se escriben a continuación del prompt, las que se indican de la forma:

**% Comando opciones parámetros**

Las opciones son modificadores para los comandos, que no siempre es necesario darlas, cada opción se representa, en la mayoría de los casos, con una sola letra precedida de un signo -, los parámetros son información que el comando necesita, lo que depende del comando en particular. Por ejemplo:

```
% wc -l carta documento ficha salida.dat
% ls -a -l -F carta documento salida.dat
% cal 1 1993
```

En el primer caso, el comando ejecutado es **wc** con la opción **l** y se le dan los parámetros **carta**, **documento**, **ficha** y **salida.dat**. En el segundo ejemplo, el comando es **ls**, se dan las opciones **a**, **l**, y **F** y los parámetros **carta**, **documento** y **salida.dat**. En la práctica se prefiere, si es posible, juntar todas las

opciones en un solo signo -, así, este ejemplo puede ser escrito de forma totalmente equivalente como:

```
% ls -alF carta documento salida.dat memorandum
```

Para el tercer ejemplo, el comando es **cal**, al cual no se le da ninguna opción y se le dan los parámetros 1 y 1993.

Para cambiar la password (palabra secreta) existe el comando **passwd**, el cual nos pide ingresar la password actual y luego la nueva dos veces, para evitar posibles errores. Este comando no requiere de opciones ni parámetros, se da de la forma:

```
% passwd
```

Luego de esto, para ingresar al sistema, habrá que escribir la nueva password.

Es muy importante que luego de terminar nuestro trabajo nos desconectemos del computador, con ello evitamos además que otra persona pueda ver nuestro trabajo, correo personal, etc., además deja el terminal disponible para que pueda ser usado por otra persona. La desconexión se hace por medio del comando **logout** o **exit**:

```
% logout
```

Luego de ésto el terminal debiera mostrar la pantalla de presentación que tenía al momento de comenzar la sesión

*Importante:* No debe apagar el terminal antes de dar el comando **logout** o **exit**, ya que con ello no se realiza la desconexión, por lo tanto, su cuenta sigue activa.

# Algunos Comandos

## Comandos Básicos

Uno de los primeros comandos que podemos probar es el comando **date**, el cual nos muestra la fecha y la hora del sistema, por ejemplo:

```
% date
Thu Oct 29 17:17:19 CDT 1992
```

En este caso el comando no necesita opciones ni parámetros. El siguiente ejemplo que veremos es el del comando **cal**, el cual nos muestra un calendario, por ejemplo:

```
% cal
   October 1992
 S  M Tu  W Th  F  S
              1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
```

```
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Pero se le pueden dar parámetros a este comando, los que son el mes y el año que queremos consultar, por ejemplo, si queremos ver el calendario de Enero de 1993 damos el comando:

```
% cal 1 1993
   January 1993
 S  M Tu  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

En adelante, para indicar que alguna opción o parámetro son opcionales lo haremos mediante los paréntesis cuadrados ( [ y ] ), es decir, lo que está entre corchetes se puede omitir, así:

```
% cal  [año]
```

Nos dice que podemos colocar o no un año determinado, de esta forma, si damos el comando:

```
% cal 2000
```

Muestra en pantalla un calendario del año 2000, y

```
% cal  [[mes] año]
```

Indica que opcionalmente se puede colocar el año, y de ser así, opcionalmente podemos colocar el mes. Este sería el formato más complicado para un comando, pero lo importante aquí es destacar que con una notación así estamos diciendo que para el comando **cal** se pueden dar 0, 1 o 2 parámetros, y que sólo al dar un año se puede especificar un cierto mes.

## Lista de archivos

Una de las operaciones más comunes es la de consultar cual es la lista de archivos dentro del directorio en el cual estamos trabajando, esto se realiza mediante el comando **ls**, por ejemplo:

```
% ls
bin           fondos        notas         unix.apunte
carta.Erika   lbs           personal      varios
```

De esta forma podemos saber los nombres de los archivos y directorios que tenemos en el directorio actual, pero como vemos, sólo nos muestra su nombre, sin distinguir entre archivos y directorios, para obtener una clara distinción entre ellos se utiliza la opción **-F**, por ejemplo:

```
% ls -F
bin/          fondos*       notas         unix.apunte
carta.Erika   lbs*          personal/     varios/
```

Con esta opción, el comando **ls** agrega al final del nombre el caracter ’/’ a los directorios, ’*’ a los

programas ejecutables (comandos) y ’@’ a los enlaces (links) simbólicos (ver comando ln).

Para obtener más información acerca de los archivos, se utiliza la opción **-l**:

```
% ls -l
drwxr-xr-x  2 jperez       512 Nov 10 20:11 bin
-rw-r--r--  1 jperez       272 Nov  4 20:57 carta.Erika
-rwxr-xr-x  1 jperez       155 Nov 10 20:10 fondos
-rwxr-xr-x  1 jperez        30 Nov 10 20:10 lbs
-rw-r--r--  1 jperez        19 Nov  4 20:56 notas
drwxr-xr-x  2 jperez       512 Nov 10 20:11 personal
-rw-rw-rw-  1 jperez     12109 Nov 10 20:15 unix.apunte
drwxr-xr-x  2 jperez       512 Nov 10 20:11 varios
```

La información que nos es presentada viene de la forma:

**permisos nl dueño tamaño fecha hora nombre**

Donde:

| | | |
|---|---|---|
| **permisos** | : | Corresponde a los permisos del archivo respectivo, los que se estudiarán más adelante. |
| **nl** | : | Número de links a ese archivo ( ver comando **ln** ). |
| **dueño** | : | Corresponde al userid del usuario que creó el archivo. |
| **tamaño** | : | Tamaño del archivo en Bytes ( número de caracteres ). |
| **fecha y hora** | : | Corresponde a la fecha y hora de la última modificación del archivo. |

Otras opciones del comando **ls** son:

| | | |
|---|---|---|
| **-a** | : | Muestra todos los archivos, sin esta opción, los archivos cuyos nombres comiencen con el caracter ’.’ no serán mostrados. |
| **-t** | : | Muestra los archivos ordenados por fecha de creación/modificación. |
| **-r** | : | Invierte el orden de despliegue. |
| **-R** | : | Lista los archivos del directorio y de todos los directorios dentro de él. |

Otra característica que no se ha mencionado es que el comando **ls** puede recibir como parámetro algún directorio o patrón de archivos y/o directorios, así por ejemplo, si queremos listar todos aquellos archivos que comiencen con la letra **c** se puede dar el comando:

```
% ls c*
carta.Erika
```

Para ver los archivos que tenemos en el directorio **personal** se puede dar el comando:

```
% ls personal
articulo.cristian    memo.jefe_personal    compras.mes
```

## Manejo de directorios

El usuario puede definir el directorio en el cual quiere trabajar, a dicho directorio se le llama **directorio de trabajo**, con ello, las referencias que hagamos a los archivos sin dar el camino desde la raíz son buscados en el directorio de trabajo. Cuando un usuario se conecta a su cuenta, el directorio de trabajo se define como su directorio.

Podemos averiguar en que directorio se está trabajando mediante el comando:

```
% pwd
/usr/grupo1/jperez
```

Recordemos que el símbolo '/' al comienzo del camino señala que el directorio está referenciado a partir de la raíz del disco.

## Cambio de directorio

Para cambiar el directorio de trabajo se hace mediante el comando **cd** y se da como argumento el nombre del directorio al cual queremos ir, si no se le especifica un directorio va al directorio que tiene definido en su cuenta al momento de partir (**Home**), por ejemplo:

| | | |
|---|---|---|
| % cd personal | : | Va al directorio **personal**. |
| % cd .. | : | Se devuelve al directorio anterior (padre). |
| % cd . | : | Nos cambia al directorio en que estamos trabajando, es decir, no hace nada. |

La referencia de los directorios puede ser más compleja, por ejemplo, si estamos trabajando en nuestro directorio de inicio **Home** y queremos pasar al directorio **notas** que está dentro del directorio **personal** podemos hacerlo con un solo comando **cd** de la forma:

```
% cd personal/notas
```

Este comando quiere decir que a partir del directorio de trabajo (ya que el nombre del directorio no comienza con '/') pasa al directorio **personal** y de allí pasa inmediatamente al directorio **notas**. En forma análoga, suponiendo que el directorio **personal** tiene un directorio **correo**, podemos pasar al directorio "hermano" con el comando:

```
% cd ../correo
```

Con esto le indicamos que a partir del directorio de trabajo se devuelva al directorio anterior (**personal**, suponiendo que estabamos trabajando en el directorio **notas**) y a partir de éste pasa al directorio **correo**.

## Crear y borrar directorios

La creación de directorios se hace por medio del comando **mkdir** al cual se le da como parámetro el nombre del directorio a crear, por ejemplo, para crear el directorio **varios** dentro del directorio en que estamos trabajando, se da el comando:

```
% mkdir varios
```

Similarmente, para borrar directorios existe el comando **rmdir**, dándole como argumento el nombre del directorio que queremos borrar, ello con algunas restricciones:

- No podemos borrar el directorio en el cual estamos trabajando.
- El directorio a borrar debe estar vacío, es decir, no debe contener ningún archivo ni directorio.

Se acostumbra estar en el directorio padre del directorio a borrar, así, para borrar el directorio **varios**, suponiendo que estamos trabajando en el directorio padre de éste, se debe dar el comando:

```
% rmdir varios
```

# Redireccionamientos y pipes

Cada programa que nosotros ejecutamos puede utilizar el teclado, si necesita leer alguna información, y la pantalla, si necesita escribir, ellos son conocidos como la entrada standard (*stdin*) y la salida standard (*stdout*) respectivamente. Además, ellos pueden ser redefinidos, es decir, podemos indicar a un programa que las lecturas que hacía del teclado ahora las haga de algún archivo, así mismo, la salida que era dirigida a la pantalla puede ser redirigida a otro archivo, incluso a otro programa. Debe notarse además que Unix interpreta los dispositivos como si fueran simples archivos (todos ellos en el directorio /dev), por lo que las redirecciones de entrada y salida pueden ser hechas de la misma forma a dispositivos.

La forma de realizar las redirecciones es mediante el uso de los caracteres:

< : Redirecciona la entrada leyendo desde un archivo.
> : Redirecciona la salida escribiendo en un archivo.
| : Conecta la salida del primer programa con la entrada del siguiente.

Así, si quisieramos guardar la salida del comando **ls -al** en el archivo **salida.ls** se realiza de la forma:

```
% ls -al >salida.ls
```

Si tuvieramos el comando **prom**, que calcule el promedio de una lista de números que son ingresados por teclado, y contamos con el archivo **numeros.dat** que contiene una lista de números, podríamos calcular su promedio con el comando:

```
% prom < numeros.dat
4.5
```

Notemos que la salida de dicho programa es mostrada en la pantalla, por lo que también podría ser redireccionada, por ejemplo:

```
% prom < numeros.dat >promedio
```

De esta forma, el programa **prom** lee sus datos del archivo **numeros.dat** y su salida es guardada en el archivo **promedio**. Notemos que el orden en que se especifiquen la entrada y la salida es irrelevante, lo importante es que el archivo de salida esté luego del caracter > y el de entrada luego del caracter < sin importar el número de espacios entre ellos, entonces, el ejemplo anterior es totalmente equivalente con:

```
% prom    >promedio  <  numeros.dat
```

El caracter | redirecciona la salida del comando de la izquierda con la entrada del comando de la derecha, si por ejemplo, quisieramos ver la lista de todos los usuarios conectados ordenada alfabéticamente, necesitaremos el comando **who** que muestra la lista de usuarios en la pantalla, y el comando **sort**, que ordena su entrada, y damos los comandos de la forma:

```
% who | sort
ajara    tt03    Nov 14 15:44
cdiaz    tt01    Nov 14 10:09
jperez   tt07    Nov 14 12:25
rsalas   tt02    Nov 14 09:35
tvera    tt05    Nov 14 16:45
```

Si además quisieramos guardar esta lista en el archivo **usuarios**, lo podemos indicar de la forma:

```
% who | sort >usuarios
```

Cabe destacar que cuando hacemos una redirección a un archivo, si éste no existía, entonces se crea, en caso de existir, se borra y se crea un archivo con la nueva información. Si queremos dejar la salida de un programa al final de un archivo que ya existe se hace de la forma:

```
% comando >>archivo.existente
```

Los caracteres >> agregan la salida de **comando** al final del archivo **archivo.existente**, si este archivo no existía entonces se crea.

# Permisos

Cada archivo y directorio tienen asociados diferentes permisos, estos permisos pueden ser de lectura, escritura y ejecución, y pueden ser dados sobre tres universos de usuarios: el dueño (quien creó el archivo), el grupo (aquellos usuarios que fueron definidos en el mismo grupo que el dueño del archivo) y el resto de los usuarios. Estos permisos se pueden ver con el comando **ls** y la opcion **-l**.

```
% ls -l
total 31
drwxr-xr-x  2 jperez        512 Nov 10 20:11 bin
-rw-rw-rw-  1 jperez        315 Nov 10 20:36 carta.Erika
-rwxr-xr-x  1 jperez        155 Nov 12 14:00 fondos
-rwxr-xr-x  1 jperez         30 Nov 10 20:10 lbs
-rw-r--r--  1 jperez         19 Nov  4 20:56 notas
drwxr-xr-x  2 jperez        512 Nov 10 20:11 personal
-rw-rw-rw-  1 jperez      23800 Nov 14 16:41 unix.apunte
drwxr-xr-x  2 jperez        512 Nov 10 21:22 varios
```

El conjunto de letras que aparece a la izquierda corresponde a los permisos, cada letra tiene un cierto significado.

r : lectura

w : escritura

x : ejecución

La primera letra señala si el archivo es un directorio (**d**) o no (**-**), el siguiente grupo de tres caracteres corresponde a los permisos para el dueño del archivo, los siguientes 3 a los del grupo y los últimos 3 a los del resto de los usuarios.

| *d* | *rwx* | *rwx* | *rwx* |
|---|---|---|---|
| **Directorio** | **Dueño** | **Grupo** | **Resto de Usuarios** |

Cuando en lugar de la letra correspondiente aparece un signo '-' significa que dicho permiso no está asignado, así por ejemplo, el archivo **unix.apunte** tiene permisos de lectura y escritura para el dueño, el grupo y el resto de los usuarios, por su parte, el archivo **notas** puede ser leído y escrito por el dueño y sólo leído por el grupo y el resto de los usuarios.

El cambio de estos permisos es hecho mediante el comando **chmod** el cual tiene un formato de la forma:

**chmod [quienes] que cuales archivo1 archivo2 ...**

**quienes**

**u :** usuario (dueño)

**g :** grupo

**o :** otros

**a :** todos (usuario, grupo y otros)

**que**

**+ :** agregar permisos

**- :** quitar permisos

**= :** asignar permisos

**cuales**

**r :** lectura

**w :** escritura

**x :** ejecución

Las indicaciones de **quienes**, **que** y **cuales** permisos deben ser dadas en un sólo parámetro (sin espacios entre ellos), así, si quisiéramos quitar el permiso de escritura para el grupo y otros usuarios al archivo **unix.apunte** se puede hacer con el comando:

```
% chmod go-w unix.apunte
```

Si quisieramos dejar todos los archivos (y directorios) con permiso de lectura y escritura para el grupo se daría el comando:

```
% chmod g=rw *
```

Similarmente, si queremos agregarle permiso de ejecución para el dueño al archivo **comandos** se puede dar el comando:

```
% chmod u+x comandos
```

En caso que la indicación de quienes sea omitida, se asume que se refiere a todos, por lo que, si queremos dejar el archivo **unix.apunte** sólo con permiso de lectura para todos los usuarios, basta dar el comando:

```
% chmod =r unix.apunte
```

# Procesos

Como bien se dijo al comienzo, Unix es un sistema operativo multi-tarea, es decir, que cada usuario puede estar ejecutando varios comandos a la vez, esto es útil sobre todo cuando se tienen programas que demoran mucho en ejecutar (como una simulación o la obtención de resultados por métodos númericos). A cada programa que se está ejecutando en el computador se le llama **Proceso**. Hasta el momento, en cada comando que hemos visto, Unix espera que termine para ejecutar el siguiente. Es posible indicar a Unix que luego del comando, sin importar que no haya terminado, despliegue el prompt para poder ejecutar otro comando más (en forma simultánea), esto se hace colocando el caracter **'&'** (Ampersand) al final del comando, así por ejemplo, si tenemos el programa **calculo_largo** y no queremos esperar hasta que termine para poder ejecutar otros comandos, podemos dar dicho comando de la forma:

```
% calculo_largo &
```

Pero probablemente este programa escribe sus resultados en la pantalla, lo que interfiere con los siguientes comandos que ejecutemos (sólo en su presentación en pantalla), por lo que se recomienda, en este caso, redireccionar la salida del programa:

```
% calculo_largo >salida.calculo &
```

Para consultar sobre los procesos que tenemos ejecutando en nuestra cuenta podemos utilizar el comando **ps**:

```
% ps
PID TT STAT  TIME COMMAND
140 00 IW    0:00 -csh
139 01 IW    0:02 -csh
157 01 S     0:00 vi unix.apunte
163 02 S     0:03 calculo_largo
171 01 R     0:00 ps
141 02 IW    0:00 -csh
```

En la primera columna (**PID**) nos muestra un número identificador del proceso, en la columna **TT** se señala el terminal que está ocupando dicho comando (en este caso se está ocupando más de un terminal), la columna **STAT** nos muestra el estado de la ejecución, la columna **TIME** muestra el tiempo de microprocesador que ha utilizado y finalmente el comando que estamos ejecutando.

Tal vez estemos ejecutando un programa que demora mucho y hemos olvidado colocar el caracter **'&'** al final de la línea, podemos recuperar el control suspendiendo dicho proceso (detener temporalmente su ejecución) pulsando las teclas **Control** (Ctrl) y sin soltarla pulsar la tecla **Z**, con ello, la ejecución de

nuestro programa queda detenida y nos aparece el prompt, en él podemos enviar una señal avisándole al proceso que queremos que se siga ejecutando, estas señales son enviadas por el comando **kill** de la forma:

```
% kill -(señal) pid1 pid2 ...
```

Para este caso enviaremos la señal **CONT** (continuar), suponiendo que el número del proceso (**PID**) era 163, el comando sería:

```
% kill -CONT 163
```

Otro caso común es cuando ejecutamos un comando por error y está demorando demasiado, entonces queremos eliminarlo, generalmente sirve pulsar las teclas **Control** y **C**, lo que envía una señal de interrupción (**INT**) al proceso que se está ejecutando, si esto falla habrá que enviarle una señal terminar (**TERM**) desde otro shell (una conexión a la misma cuenta desde otro terminal, por ejemplo), la señal **TERM** es la que el comando kill utiliza cuando no se le dice que señal enviar, por lo que bastaría con el comando:

```
% kill 163
```

Si esto también fallara, tenemos un último recurso, la señal destruir (**KILL**):

```
% kill -KILL 163
```

Las versiones más antiguas de Unix no aceptan la indicación de la señal con letras, por lo que debe ser indicada con números según la siguiente tabla:

STOP = 17
CONT = 19
TERM = 15
INT = 2
KILL = 9

Con esto: kill -KILL 163 es equivalente a kill -9 163.

Esta labor parece bastante engorrosa, pero si usted dispone del shell C-Shell (csh) esta labor se le facilitará. Vea la sección de C-Shell para más detalle.

# Ayuda

Unix nos provee de una ayuda que, por lo general, es bastante completa, y con ella podemos aprender muchos comandos de Unix sin necesidad de tener un pesado manual a mano. Si queremos saber más sobre el comando **ls** podemos hacerlo mediante el comando:

```
% man ls
```

El comando que nos proporciona la ayuda es el comando **man**, al cual se le da como argumento el

tópico sobre el cual queremos averiguar. Esta ayuda la muestra por medio del comando **more** (ver comando **more**), puede obtener una ayuda rápida de él pulsando la tecla h.

También podemos encontrar una lista de los tópicos relacionados con una palabra por medio del comando **apropos** dándole como parámetro la palabra sobre la cual queremos encontrar información, por ejemplo:

```
% apropos latex
latex (1)    - structured text formatting and typesetting
slitex (1)   - make LaTeX slides
```

El número que muestra entre paréntesis corresponde a la sección, en caso de haber ambigüedad ( el mismo tópico en dos secciones distintas), se puede dar la sección correspondiente al comando man, por ejemplo:

```
% man 1 latex
```

# El editor VI

La forma más cómoda de trabajar con un texto ( modificarlo ) es mediante un editor, Unix nos provee del editor **vi**, aunque su manejo es un tanto complejo, permite realizar operaciones que no todos los editores permiten.

La forma de invocarlo es:

```
% vi archivo
```

La mayoría de los comandos se dan pulsando las teclas indicadas sin que éstas aparezcan en la pantalla ni tampoco es necesario pulsar **<ENTER>** al final de ellos, sólo los comandos que comienzan con :,/ y ? son mostrados en la última línea de la pantalla y requieren la pulsación de <ENTER> para finalizar (estos corresponden a los comandos del editor **ex**, en el cual se basa **vi**).

Antes de comenzar a describir los comandos se establecerán las normas de la notación:

| i | *texto* | < ESC > |
|---|---|---|
| **(1)** | **(2)** | **(3)** |

1. Corresponde al comando, en este caso basta pulsar sólo la letra i
2. Aquí debe ingresar el texto que desee, puede utilizar más de una línea pulsando <ENTER> al final de cada una
3. Debe pulsar la tecla <ESC> (Escape)

| | | |
|---|---|---|
| *c* | : | Un caracter cualquiera |
| *l* | : | Una letra del alfabeto inglés |
| ^X | : | Pulsar las teclas <CONTROL> y X |
| caracter | : | Un caracter cualquiera |

| | | |
|---|---|---|
| CARACTER | : | Un caracter distinto de espacio |
| palabra | : | Una secuencia de letras y/o números |
| PALABRA | : | Una secuencia de caracteres incluyendo los espacios que siguen |
| *arch* | : | Algún archivo del disco (Existente o no) |
| *patrón* | : | Secuencia de caracteres a utilizar en un patrón de búsqueda |
| *movimiento* | : | Algún comando de movimiento |

A continuación se describirán la mayoría de los comandos de vi (sólo se excluyen los más complicados), a muchos de éstos se les puede anteponer un número decimal que indica un factor de repetición del comando, es decir, si se escribe 20 antes del comando, éste se repite 20 veces. Los comandos que tienen esta capacidad serán señalados con una letra *n* en la columna izquierda, al no colocar el valor, el comando se ejecuta sólo una vez.

## Comandos de Movimiento

| *n* | Comando | Descripción |
|---|---|---|
| *n* | l ó → | *n* caracteres a la derecha |
| *n* | h ó ← | *n* caracteres a la izquierda |
| *n* | k ó ↑ | *n* líneas arriba |
| *n* | j ó ↓ | *n* líneas abajo |
| -- | ^F | Avanza una página |
| -- | ^B | Retrocede una página |
| -- | ^D | Avanza media página |
| -- | ^U | Retrocede media página |
| -- | ^U | Retrocede media página |
| *n* | $ | Avanza hasta el final de *n*-1 líneas adelante |
| -- | ^ | Va al primer caracter distinto de espacio de la línea |
| *n* | _ | Va al primer caracter distinto de espacio de *n*-1 líneas adelante |
| *n* | - | Va al primer caracter distinto de espacio *n* líneas atrás |
| *n* | + ó <ENTER> | Va al primer caracter distinto de espacio *n* líneas adelante |
| -- | 0 (Cero) | Va al primer caracter de la línea (incluso espacio) |
| *n* | \| | Va a la columna *n* dentro de la línea |
| *n* | f*c* | Avanza hasta el caracter *c* |
| *n* | t*c* | Avanza hasta la posición anterior al caracter *c* |
| *n* | F*c* | Retrocede hasta el caracter *c* |
| *n* | T*c* | Retrocede hasta la posición siguiente al caracter *c* |
| *n* | ; | Repite el último comando ’f’, ’t’, ’F’ o ’T’ |
| *n* | , | Idéntico al anterior, pero en la dirección opuesta |

| | | |
|---|---|---|
| *n* | w | Avanza *n* palabras |
| *n* | W | Avanza *n* PALABRAS |
| *n* | b | Retrocede *n* palabras |
| *n* | B | Retrocede *n* PALABRAS |
| *n* | e | Avanza al final de *n* palabras adelante |
| *n* | E | Avanza al final de *n* PALABRAS adelante |
| *n* | G | Va a la línea *n* (última si *n* no se especifica) |
| *n* | H | Va a la línea *n* a partir de la primera que se ve en pantalla |
| *n* | L | Va a *n*-ésima línea anterior a la última que se ve en pantalla |
| -- | M | Va a la línea del medio de la pantalla |
| *n* | ) | Avanza *n* sentencias |
| *n* | ( | Retrocede *n* sentencias |
| *n* | { | Avanza *n* párrafos |
| *n* | } | Retrocede *n* párrafos |
| -- | ‘*l* | Va a la marca *l* |
| -- | ’*l* | Va al primer CARACTER dentro de la línea con la marca *l* |
| -- | ‘‘ | Va a la posición anterior al último salto |
| -- | ’’ | Va al primer CARACTER dentro de la línea en que se encontraba el cursor antes del último salto |
| -- | /*patrón* | Avanza hasta la siguiente ocurrencia del *patrón* |
| -- | ?*patrón* | Retrocede a la anterior ocurrencia del *patrón* |
| -- | n | Repite el último comando ’/’ o ’?’ |
| -- | N | Igual al anterior, pero en la dirección opuesta |
| -- | % | Busca el siguiente paréntesis o su pareja (también con {, }, [ y ]) |

## Comandos de Inserción de texto

| *n* | Comando | Descripción |
|---|---|---|
| *n* | i*texto*<ESC> | Inserta *texto* en la posición actual del cursor |
| *n* | a*texto*<ESC> | Agrega *texto* en la posición siguiente a la del cursor |
| *n* | I*texto*<ESC> | Inserta *texto* delante del primer CARACTER de la línea actual |
| *n* | A*texto*<ESC> | Agrega *texto* al final de la línea actual |
| *n* | o*texto*<ESC> | Agrega *texto* en la línea siguiente |
| *n* | O*texto*<ESC> | Agrega *texto* en la línea anterior |
| *n* | p | Coloca el último grupo de líneas guardado o borrado en la línea siguiente *n* veces |
| *n* | P | Coloca el último grupo de líneas guardado o borrado en la línea anterior *n* veces |
| *n* | . | Repite el último comando *n* veces |

## Comandos de Reemplazo

| *n* | Comando | Descripción |
|---|---|---|
| *n* | r*c* | Reemplaza *n* caracteres por *c* |
| *n* | R*texto*<ESC> | Sobreescribe el resto de la línea, agregando *n*-1 veces |
| *n* | s | Sustituye *n* caracteres |
| *n* | S | Sustituye *n* líneas |
| *n* | c*movimiento texto*<ESC> | Cambia lo alcanzado por *n movimientos* por *texto* |
| *n* | cc*texto*<ESC> | Cambia *n* líneas por *texto* |
| *n* | C*texto*<ESC> | Cambia el resto de la línea las y *n*-1 líneas siguientes por *texto* |
| -- | ~ | Intercambia entre mayúsculas y minúsculas |
| *n* | J | Junta *n* líneas (Si *n* no se especifica se junta la actual con la siguiente) |
| *n* | . | Repite el último comando *n* veces (J sólo una vez) |
| -- | :[x,y]s/*patrón*/*texto*/*m* | Sustituye el texto alcanzado por *patrón* por *texto* entre las líneas x e y (% para señalar todas). El modificador *m* puede ser g (Global) o c (Con confirmación) |
| -- | & | Repite el último reemplazo dado con el comando anterior |

## Comandos de Borrado

| *n* | Comando | Descripción |
|---|---|---|
| *n* | x | Borra *n* caracteres a partir de la posición del cursor |
| *n* | X | Borra *n* caracteres antes del cursor |
| *n* | d*movimiento* | < Borra *n* veces lo indicado por *movimiento* (3dw’ Borra 3 palabras) |
| *n* | dd | Borra *n* líneas |
| -- | D | Borra hasta el final de la línea |
| *n* | . | Repite el último comando *n* veces |

## Comandos de Copia y Marcado

| *n* | Comando | Descripción |
|---|---|---|
| *n* | y*movimiento* | Marca el texto descrito por *movimiento* para copiarlo con el comando p o P |
| *n* | yy | Marca *n* líneas para copiarlas con el comando p o P |
| *n* | Y | Marca *n* líneas para copiarlas con el comando p o P |
| -- | m*l* | Marca la posición del cursor con la letra *l* |

## Comandos para deshacer

| Comando | Descripción |
|---|---|
| u | Deshace la última modificación |
| U | Deshace todos los cambios hechos en la línea actual |
| p | Coloca el último grupo de líneas guardado o borrado en la línea siguiente |
| P | Coloca el último grupo de líneas guardado o borrado en la línea anterior |
| :q! | Abandona vi sin grabar las modificaciones |
| :e! | Re-edita el archivo (Como salir y editarlo nuevamente) |

### Comandos de Grabación y Salida

| Comando | Descripción |
|---|---|
| :q | Sale de vi (Si no se ha modificado desde la última grabación) |
| :q! | Sale sin grabar |
| :w | Graba el archivo |
| :w *arch* | Graba en el archivo *arch* |
| :w >> *arch* | Agrega el archivo editado al archivo de nombre *arch* |
| :w! *arch* | Graba el archivo editado con nombre *arch* sin importar que éste exista |
| :*x*,*y* w *arch* | Graba de la línea *x* a la *y* en el archivo *arch* |
| :wq | Graba y sale |
| :ZZ | Graba slo si el archivo ha sido modificado y sale |
| :f *arch* | Cambia el nombre del archivo editado a *arch* |
| :r *arch* | Agrega el archivo *arch* después de la línea actual |

# C-Shell (csh)

C-Shell es un intérprete de comandos (Shell), más poderoso que el standard **Bourne Shell** provisto con Unix, el cual se está convirtiendo en el nuevo standard.

En este capítulo se comentarán algunas de las facilidades proporcionadas por C-Shell, tales como: History, que es una forma cómoda de repetir comandos realizados anteriormente; alias, una forma alternativa de nombrar a los comandos y Jobs, una forma más fácil de manejar procesos, pero un tanto más restringida.

## History

Esta capacidad que incorpora C-Shell nos permite usar palabras de los comandos escritos previamente, en este capítulo veremos sólo su uso más sencillo que consiste en repetir algún comando.

Podemos ver la lista de comandos que "recuerda" C-Shell dando el comando:

```
% history
1 ls -F
2 cat carta.Erika
3 rm reunion.memo
4 cd apuntes
5 vi unix.txt
```

Se dispone de dos formas para repetir alguno de estos comandos, la primera requiere que demos el comando **history** y nos fijemos en el número que aparece a la izquierda del comando que deseamos repetir. Para repetir el comando **cd apuntes** basta el comando de repetición de la forma:

```
% !3
```

La segunda alternativa es más amigable, sólo necesita las primeras letras del comando, buscando hacia atrás el primer comando que coincida con ellas, así, para repetir el comando **cd apuntes** se puede indicar de la forma:

```
% !cd
```

O equivalentemente:

```
% !c
```

Ya que es el último comando realizado que comienza con la letra ’c’.

## Alias

Alias es una forma de definir sinónimos a los comandos del sistema o incluso redefinir los existentes con alguna forma más complicada (definirlos con otro comando es posible, pero no es gracioso cuando otra persona define el comando **cp** como **rm** sin advertirlo).

Si queremos definir un nuevo comando que borre archivos, por ejemplo: **borrar** lo podemos hacer de la forma:

```
% alias borrar rm
```

Ya que **rm** es el comando Unix para borrar archivos (Ver Apéndice A para más detalles).

Un caso muy utilizado es el de redifinir **ls** por **ls -F**.

```
% alias ls ls -F
```

Con ello, cada vez que se utilice el comando **ls** se entenderá como **ls -F**

Se puede obtener una lista de los alias definidos dando el comando **alias** sin parámetros:

```
% alias
```

Para borrar alguna definición de alias, se hace por medio del comando **unalias**, especificando el alias que deseamos eliminar:

```
% unalias borrar
```

Borrará el alias **borrar**.

## Jobs

Esta es una forma más práctica que nos provee C-Shell para el manejo de procesos, pero sólo de aquellos ejecutados en dicho C-Shell.

Para consultar los procesos que tenemos en dicho C-Shell se debe dar el comando:

```
% jobs
[1]  +    Running                 ls -al
[2]       Running                 calculo_largo
[3]  -    Running                 who
```

Nos muestra a la izquierda (entre paréntesis de corchetes) un número de identificación del trabajo, para referenciar alguno de ellos se debe colocar dicho número precedido del signo %, también pueden ser referenciados con un signo % seguido de las primeras letras del comando (al igual que con el comando **history**, salvo que en este caso se busca hacia adelante), además pueden distinguirse un proceso marcado con el signo '+', el cual se utiliza en caso de omitir la identificación del proceso o puede especificarse como %+ o %%; y otro marcado con el signo '-', referenciado como %-.

Ahora veremos una lista de los comandos de C-Shell con respecto al manejo de procesos:

| | | |
|---|---|---|
| jobs [-l] | : | Muestra la lista de procesos del C-Shell en curso, al dar la opción **-l** muestra además el **PID** (El mismo número identificador de proceso mostrado en **ps**). |
| stop [%*job*] | : | Detiene el proceso *job* |
| fg [%*job*] | : | Continua la ejecución del proceso *job*, pero espera hasta que éste termine (ejecución foreground). |
| %*job* | : | Equivalente a **fg** *job* |
| bg [%*job*] | : | Continua la ejecución del proceso *job* dejando al C-Shell disponible inmediatamente para dar otros comandos (ejecución background). |
| %*job* & | : | Equivalente a **bg** *job* |
| kill [-*señal*] [%*job*] | : | Envía una señal al proceso *job*, de la misma forma explicada en Procesos, salvo que la especificación del proceso puede ser realizada por esta forma sencilla provista por C-Shell. |

Así, si queremos detener el proceso ls -al >salida, puede ser realizado de cualquiera de las siguientes formas:

- % stop
- % stop %%
- % stop %+
- % stop %1
- % stop %ls

Por otra parte, si queremos destruir el proceso calculo_largo >salida.calculo_largo, puede ser hecho de

las formas:

- % kill -KILL %2
- % kill -KILL %calc
- % kill -KILL %c

## Archivos de Inicialización

Hay muchas definiciones que se deben realizar cada vez que un usuario se conecta a su cuenta, y otras tantas que se prefiere hacerlas en cada conexión (no obligatorias, pero cómodas), como por ejemplo algunos **alias**.

Cada usuario cuenta al menos con dos de estos archivos de inicializaciones, ellos son:

- .login : Se ejecutan los comandos que contiene cada vez que el usuario se conecta.
- .cshrc : Se ejecutan los comandos que contiene cada vez que se llama al C-Shell (al momento de conectarse y cada vez que se da el comando **csh**, para salir de él se da el comando **exit**.

Note que los nombres de estos archivos comienzan con el caracter '.', ya que no es necesario que se listen en cada comando **ls** solicitado. Para obligar que sus nombres sean listados, recuerde dar la opción **-a** al comando **ls**.

> *Advertencia:* *No modifique estos archivos a menos que esté seguro de lo que está haciendo. Como forma de prueba puede copiar el archivo con otro nombre antes de modificarlo, con el fin de poder recuperar las definiciones anteriores.*

Podemos editar estos archivos con el editor **vi**, por ejemplo:

```
% vi .login
```

En él puede agregar los **alias** que estime conveniente, si ya tiene algunos definidos es recomendable dejarlos todos juntos, solo con fines de órden.

También se pueden agregar algunas **variables de ambiente**, con lo que se puede definir el modo de trabajo dentro del C-Shell, estas modificaciones se hacen por medio del comando **set**, de la forma:

```
% set variable [= valor]
```

Estas definiciones también pueden ser hacer dentro de la sesión (en la línea de comandos), pero sólo tendrán validez dentro de dicha sesión.

Para borrar alguna definición, se utiliza el comando **unset** especificando la variable que se desea borrar, de la forma:

```
% unset variable
```

podemos ver la lista de variables definidas dando el comando **set** sin parámetros:

```
% set
```

Algunas de estas variables se explican a continuación:

| | | |
|---|---|---|
| history | : | Número de comandos a guardar en la lista de history. |
| ignoreeof | : | Evita que se dé un **logout** automático al pulsar CTRL-D. |
| noclobber | : | Evita el borrado accidental de archivos en redireccionamientos. |
| noglob | : | Los caracteres de globalización son interpretados como si mismos, sin ser reemplazados por los nombres de los archivos que cumplan en patrón. |
| notify | : | Informa el término de alguna tarea en forma asincrónica, si esta variable no está seleccionada, se informa sobre el término de los procesos sólo antes de escribir el siguiente prompt. |
| prompt | : | Mensaje que se coloca para señalar que se está en espera de un comando, cuando aparece el caracter "!", en esa posición se coloca el número que tendrá en la lista de **history**. |
| savehits | : | Número de comandos de la lista **history** que se "recordarán" en la siguiente sesión. |

Ejemplo:

```
set history=20
set ignoreeof
set noclobber
set prompt="listo para recibir comando numero ! % "
set savehist=20
```

# Apéndice

| | | |
|---|---|---|
| Comando | : | **at** |
| Descripción | : | Ejecuta un comando o un archivo de comandos en la fecha y hora indicados. |
| Sintaxis | : | **at** hora [día] [archivo] |
| Observaciones | : | |

- Para la especificación de fecha y hora, siempre se busca el momento más reciente que cumpla la especificación.
- Si no se especifica el archivo de comandos, son leídos desde el teclado, para finalizar la entrada de comandos se debe pulsar CTRL-D (se mantiene apretada la tecla Control y se presiona D).
- Los comandos dados en el archivo de comandos (o de son dados de la misma forma que en la línea de comandos. Se recomienda utilizar redireccionamientos para los resultados.

Ejemplos :

```
% at 1200
```

```
% at 2:00pm sunday  proceso_largo
% at 1000 feb 14 enviar_carta_amor
% at 0000 Jan 1 agno_nuevo
% at 1200 Apr 12, 1997 30_aniversario
% at now + 2 hour es_tarde
```

---

Comando : **cal**

Descripción : muestra un calendario del año y mes especificados.

Sintaxis : **cal** [ [mes] año ]

Observaciones :

- mes debe ser especificado como número y el año como entero de cuatro dígitos.
- Si no se especifican ni el mes ni el año, se muestra un calendario del mes actual.

Ejemplos :

```
% cal 2000  calendario del año 2000 )
% cal 3 1995  ( calendario de Marzo de 1993 )
```

---

Comando : **cat**

Descripción : muestra el contenido de los archivos especificados.

Sintaxis : **cat** [archivo1 archivo2 ...]

Observaciones :

- Los archivos son mostrados hacia la salida standard en el mismo orden que son dados en la línea de comandos.
- Si no se especifica ningún archivo, este se lee de la entrada standard.

Ejemplos :

```
% cat carta.Erika
( Muestra el contenido del archivo carta.Erika en pantalla )


% cat memo1 memo2 memo3 > memos
( Deja en el archivo memos la unión de los archivos memo1
memo2 y memo3 )


% cat >salida
( Deja en el archivo salida lo que se ingrese por teclado
hasta pulsar CTRL-D )
```

---

Comando : **cd** ( Ver Capítulo 2 )

Descripción : Cambia el directorio de trabajo

Sintaxis : **cd** [ directorio ]

Observaciones :

- Si no se especifica un directorio se cambia al directorio HOME ( directorio de inicio de la sesión ).
- Opcionalmente el directorio puede tener un signo / al final, es decir, cd varios es equivalente a cd varios/

Ejemplos :

```
   % cd notas
( Se cambia al directorio notas que se encuentra dentro
 del directorio actual )

   % cd /usr/local/public
( Se cambia al directorio /usr/local/public, el signo /
al comienzo señala que dicho directorio se especifica
a partir de la raíz )

   % cd ..
( Se cambia al directorio padre )
```

---

Comando : **chmod**

Descripción : Cambia los permisos de los archivos y/o directorios especificados

Sintaxis : **chmod** [quienes] que cuales archivos ...

**quienes**

**u :** usuario (dueño)

**g :** grupo

**o :** otros

**a :** todos (usuario, grupo y otros)

**que**

**+ :** agregar permisos

**- :** quitar permisos

**= :** asignar permisos

**cuales**

**r :** lectura

**w :** escritura

**x :** ejecución

Ejemplos :

```
   % chmod +x saludo
( Da permiso de ejecución a  saludo, para todos los
usuarios )

   % chmod go-r *
```

```
( Quita permiso de lectura a todos lo archivos del directorio
actual )

   % chmod g=rw
( Fija los permisos para el grupo en lectura/escritura )
```

---

Comando : **comm**

Descripción : compara dos archivos

Sintaxis : **comm** [opciones] archivo1 archivo2

Opciones : -1 No lista las líneas que existen sólo en archivo1

-2 No lista las líneas que existen sólo en archivo2

-3 No lista las líneas que existen en ambos archivos

Ejemplos :

```
   % comm carta.old carta
( Compara los archivos carta.old y carta )
```

---

Comando : **cp**

Descripción : Copia archivos

Sintaxis : **cp** archivo_fuente archivo_destino (1)

**cp** archivo1 archivo2 ... directorio_destino (2)

Observaciones :

(1) Copia archivo_fuente en archivo_destino

(2) Copia los archivos archivo1, archivo2, etc. en el directorio directorio_destino, conservando los mismos nombres de archivos

Ejemplos :

```
   % cp reajustes copia_reajustes

   % cp reajustes ..
( copia el archivo reajustes en el directorio padre )

   % cp reaj* reajustes_1993
( copia todos los archivos cuyo nombre comience con reaj
en el directorio reajustes_1993 )
```

---

Comando : **diff**

Descripción : muestra las diferencias entre dos archivos

Sintaxis : **diff** archivo1 archivo2

Ejemplos :

```
   % diff carta.old carta
( Muestra las diferencias entre los archivos carta.old
y carta )
```

---

Comando : **echo**

Descripción : muestra un mensaje

Sintaxis : **echo** [-n] mensaje

Observaciones : Al especificar la opción **-n**, la siguiente escritura es hecha en la misma línea

Ejemplos :

```
   % echo esto es un mensaje
( Muestra 'esto es un mensaje' en la pantalla )

   %  echo esto es una prueba de creació de archivo >sal.echo
( Crea el archivo 'sal.echo' con el contenido 'esto es una
prueba de creación de archivo' )
```

---

Comando : **expr**

Descripción : Evalúa una expresión

Sintaxis : **expr** expresión

Observaciones :

- Cada operando u operador debe estar separado por un espacio del siguiente
- Operadores :

  \+ Suma

  \- Resta

  * Multiplicación

  / División

- Para evitar que el caracter * sea interpretado como un caracter de globalización, se le antepone un \ (backslash) o se coloca entre comillas simples , igualmente con los paréntesis
- Sólo se trabaja con números enteros
- Las multiplicaciones y divisiones son realizadas antes que las sumas y restas

Ejemplos :

```
   % expr 2 + 5
   ( Muestra 7 )

   % expr 2 + 7 \* 7

   % expr \( 2 + 5 \) \* 7
   ( Muestra 49 )
```

---

Comando : **file**

Descripción : Indica la clasificación de los archivos

Sintaxis : **file** archivo1 archivo2 ...

Ejemplos :

```
% file *
News:               directory
apuntes:            directory
lbs:                commands text
bin:                directory
file-formats:       English text
finding-files:      ascii text
svga111.zip:        data
```

---

Comando : **grep**

Descripción : Muestra las líneas que cumplan con cierto patrón en los archivos especificados

Sintaxis : **grep** patrón archivo1 archivo2 ...

Ejemplos :

```
% grep dispositivo apunte.unix
( Muestra las líneas que tengan la palabra dispositivo del
archivo apunte.unix )

%  who | grep jperez
( Muestra las líneas que contengan la palabra jperez de la
lista de usuarios conectados )
```

---

Comando : **kill** ( Ver Capítulo 7 )

Descripción : Envia una señal de término a un proceso

Sintaxis : **kill** [-señal] proceso1 proceso2 ...

Ejemplos :

```
% kill 20192 20194 20201

% kill -KILL 20192 20194 20201
```

---

Comando : **ln**

Descripción : Referencia un archivo con otro nombre ( incluso en otro directorio )

Sintaxis : **ln** [-s] archivo_existente nombre_enlace

Opciones :

-s Crea un enlace (link) simbólico (utiliza menos memoria para el usuario)

Observaciones :

- Cada vez que se referencie el archivo **nombre_enlace** se utilizará el archivo **archivo_existente**
- También es válido con directorios

Ejemplos :

```
   % ln ~agonzale/carta.director carta.director
( Crea un archivo carta.director en el directorio actual y
cada vez que se referencie a éste, en realidad se utilizará
el archivo carta.director que está en la cuenta agonzale )
```

---

Comando : **lpr**

Descripción : Imprime archivos

Sintaxis : **lpr** archivo1 archivo2 ...

Ejemplos :

---

Comando : **ls** ( Ver Capítulo 4.2 )

Descripción : Muestra la lista de archivos

Sintaxis : ls [opciones] [lista de archivos y/o directorio]

Opciones :

-a Muestra todos los archivos

-l Descripción detallada

-t Muestra los archivo ordenados desde el más reciente al más antiguo

-F Identifica archivos comunes, directorios y links simbólicos

Ejemplos :

```
   % ls -lF apun*

   % ls -Ft entradas salidas estados *.err
```

---

Comando : **mail**

Descripción : Envía o recibe correo

Sintaxis : **mail** [usuario1 usuario2 ...]

Observaciones :

- Al especificar una lista de usuarios envía la carta a los usuarios especificados

- Para terminar una carta se coloca un '.' (punto) o se pulsa CTRL-D en la primera columna (luego de un < ENTER > )
- Al dar el comando mail sin parámetros se ingresa al modo de comando internos de mail, pudiendo leer las cartas recibidas.

Comandos internos :

| | |
|---|---|
| d [*n*] | Borra la carta número *n* o la actual si *n* no se especifica |
| q | Salir |
| w [*n*] [*archivo*] | Graba la carta número *n*, o la actual si *n* no se especifica, en el archivo *archivo* |
| <ENTER> | Muestra la carta actual |
| h | Muestra la lista de cartas recibidas |
| *n* | Muestra la carta número *n* |

---

Comando : **mesg** ( Ver Comando write )
Descripción : Permite o prohibe la recepción de mensajes
Sintaxis : **mesg** y ( Permite )
**mesg** n ( Prohibe )

---

Comando : **mkdir** ( Ver Capítulo 2 )
Descripción : Crea un directorio
Sintaxis : **mkdir** directorio
Ejemplos :

```
% mkdir apuntes   ( Crea el directorio apuntes )
```

---

Comando : **more**
Descripción : Muestra un archivo en forma pausada ( de a una página )
Sintaxis : **more** archivo1 archivo2 ...
Comandos internos :

| | |
|---|---|
| <ESPACIO> | Muestra página siguiente |
| b | Muestra página anterior |
| h | Muestra ayuda |
| q | Sale |

Ejemplos :

```
   % more datos.salida
( Despliega el archivo datos.salida en forma pausada )
```

---

| | | |
|---|---|---|
| Comando | : | **mv** |
| Descripción | : | Mueve (o renombra) archivos |
| Sintaxis | : | **mv** archivo_viejo archivo_nuevo (1) |
| | : | **mv** archivo1 archivo2 ... directorio (2) |
| Observaciones | : | (1) Cambia el nombre de archivo_viejo a archivo_nuevo |
| | | (2) Mueve los archivos al directorio especificado |
| Ejemplos | : | |

```
   % mv carta carta.anterior
( Cambia el nombre del archivo carta a carta.anterior )

   % mv *.doc *.txt documentos
( Mueve todos los archivos cuyo nombre termina con '.doc' o
'.txt' al directorio documentos, los nombres de los archivos
se mantienen )
```

---

| | | |
|---|---|---|
| Comando | : | **pr** |
| Descripción | : | Prepara un archivo para imprimirlo |
| Sintaxis | : | **pr** [opciones] [archivo1 archivo2 ...] |
| Opciones | : | |
| | **-h** | Coloca el texto que sigue a esta opción al comienzo de cada página |
| | **-ln** | Fija el número de líneas por página en **n** |
| | **-wn** | Fija el ancho a **n** caracteres |
| Ejemplos | : | |

```
   % pr -h 'Apunte Unix' -l66 -w70 unix.apunte | lpr
( Prepara e imprime el archivo unix.apunte )
```

---

| | | |
|---|---|---|
| Comando | : | **ps** |
| Descripción | : | Despliega el estado de los procesos |
| Sintaxis | : | **ps** [opciones] |
| Opciones | : | |
| | **-a** | Procesos de todos los usuarios |
| | **-l** | Lista detallada |

**-x** Muestra todos los procesos ( del usuario o del computador según la opcion -a )

Ejemplos :

```
   % ps -l
( Muestra una lista detallada de los procesos del usuario )

   % ps -la

( Muestra una lista detallada de los procesos de todos los
usuarios )

   % ps -x
( Muestra todos los procesos del usuario, incluso aquellos
que no tienen un terminal asignado )
```

---

Comando : **rm**

Descripción : Borra archivos

Sintaxis : **rm** [opciones] archivo1 archivo2 ...

Opciones : **-i** Pide confirmación antes de borrar

**-r** Borra un directorio y todos los archivos y directorios que están dentro de él

Ejemplos :

```
   % rm -i *
( Borra todos los archivos del directorio actual pidiendo
confirmación antes de borrar )

   % rm *.c ?[aeiou]*

( Borra todos los archivos terminados en '.c' y aquellos
que su segunda letra es una vocal )
```

---

Comando : **rmdir** ( Ver Capítulo 4 )

Descripción : Borra directorios

Sintaxis : **rmdir** directorio1 directorio2 ...

Ejemplos :

```
   %  rmdir vacio
( Borra el directorio de nombre vacio )
```

---

Comando : **sort**

Descripción : ordena archivos

Sintaxis : **sort** [opciones] [campos] [archivo1 archivo2 ...]

Opciones :

**-b** Ignora blanco al principio y al final de la línea

**-f** Considera mayúsculas y minúsculas iguales

**-n** Orden numérico

**-r** Invierte el orden

**-tx** El caracter x se considera como separador de campos

Observaciones :

- La indicación de campos señala bajo que campos se desea ordenar, ellos se especifican de la forma {**+n** y **-m** donde **n** es el número del primer campo a ordenar y **m** el del último. Los campos son numerados a partir de 0.
- Si no se indica un archivo de entrada los datos son leídos de la entrada standard

Ejemplos :

```
   % who | sort
( Muestra la lista de usuarios ordenada alfabéticamente )

   % sort -n resultados
( Muestra las líneas del archivo resultados ordenadas
numéricamente )

   % sort -n +2 resultados
( Muestra las líneas del archivo resultados ordenadas
numéricamente según el tercer campo del archivo )

   % sort -n +2 +0
( Muestra las líneas del archivo resultados ordenadas
numéricamente según el tercer campo del archivo, y en caso de
igualdad, según el primer campo )
```

---

Comando : **wc**

Descripción : Informa el número de líneas, palabras y/o caracteres de los archivos

Sintaxis : **wc** [opciones] archivo1 archivo2 ...

Opciones :

**-c** desplegar el número de caracteres

**-l** Desplegar el número de líneas

**-w** Desplegar el número de palabras

Observaciones : - Si no se dan opciones se asume **-lwc**

- Si no se especifican archivos se lee de la entrada standard

Ejemplos :

```
   % wc *
( Muestra el número de líneas, palabras y caracteres
de todos los archivos del directorio actual )
```

---

Comando : **who**

Descripción : Muestra la lista de usuarios conectados

Sintaxis : **who** [am i]

Observaciones : Al dar los parámetros **am i** muestra la información de la conexión en curso, si no, muestra la lista de todos los usuarios conectados.

Ejemplos :

```
   % who | wc -l
( Muestra el número de usuarios conectados )
```

---

Comando : **write**

Descripción : Envía un mensaje a otro usuario

Sintaxis : **write** usuario [tty]

Observaciones :

- El mensaje es leído de la entrada standard, si se está utilizando el teclado, la conversación se finaliza pulsando **CTRL-D**
- tty se refiere al terminal al cual quiere establecer la comunicación en caso que el usuario esté en más de un terminal
- Sólo es posible enviar un mensaje si el destinatario lo permite ( Ver Comando mesg )

Ejemplos :

| Usuario jperez | Usuario agonzale |
|---|---|
| % write agonzale | |
| | Message from jperez on tty02 at 12:27 ...<br>% write jperez tty02 |
| Message from agonzale on tty13 at 12:28 | |
| Hola Alejandro<br>Que te parece si vamos a almorzar juntos? | |
| | Me parece bien, termino en 15 minutos<br>Nos encontramos en la entrada |
| De acuerdo<br>(CTRL-D) | |
| | EOF<br>(CTRL-D) |
| EOF | |

---

*Centro de Computación, Universidad de Chile*