Apunte Unix Avanzado

Centro de Computación

Facultad de Ciencias Físicas y Matemáticas

Universidad de Chile

Julio de 1998

Versión Revisada

Indice General

1. ¿Qué es un sistema operativo?

Una definición un tanto descriptiva es que un sistema operativo *es un gran cuerpo formado por software que actúa como un policía de tránsito que dirige el flujo de información de y hacia el hardware*. Básicamente un sistema operativo como Unix se divide en las siguientes partes:

- Controladores de dispositivos, entiéndase por dispositivos los elementos del hardware que son controlados por el sistema operativo.
- Un *scheduler*, que es el que decide que programa de usuario se ejecutará, cuál y por cuanto tiempo se ejecutará.
- Manejador de memoria que determina cuanta memoria es entregada para cada programa y soluciona problemas derivados de esta actividad, como que hacer si falta memoria, etc.
- Sistema de archivos que se preocupa de localizar, manejar y administrar los archivos en disco.
- Sistema de programas. Este es un grupo de programas provistos por el sistema operativo.

1.1 Definiciones

1.1.1 Shell

Shell es un programa que procesa comandos para que el usuario interactue con el sistema operativo. En el caso de Unix permite la interacción con el ***Kernel.***

En Unix cuando está presente el ***shell*** se distingue por el *prompt* (o línea de comandos) que para este apunte lo notaremos simplemente con el símbolo **%>**. Otra característica importante es que permite la programación de programas de comandos (o *shell scripts*). Existen muchos ***Shell*** para Unix siendo el más estandard el **Bournel Shell** (**sh**), otros son el ***TC Shell*** y ***C Shell.***

1.1.2 Kernel

El ***Kernel*** de Unix es el que realiza todas las labores propias de un sistema operativo que

quedan al excluir el ***Shell***, estas son: la de manejo de memoria, control de dispositivos, scheduler, etc. Este es el corazón del sistema ya que además posee un gran número de funciones de sistemas que son invisibles para el usuario casual (se llaman *system calls*). Es cargado por el computador al encenderse y funciona hasta que este es apagado o se hace un *shutdown* (bajar el sistema).

1.1.3 Dispositivos Estandares

Existen en Unix tres dispositivos estandares para los comandos, estos son: **La Entrada Estándar**, **La Salida Estándar** y **La Salida de Error Estándar**. Estos dispositivos pueden ser redireccionados a/o desde archivos (dependiendo si son de entrada o de salida).

- **Salida estándar**

  Se le llama **salida estándar** al dispositivo en que el comando arroja sus resultados, comúnmente es la pantalla, a excepción que se especifique de otro modo. Existen comandos que al no especificárseles argumentos o un archivo de salida utilizan la salida estándar.

- **Salida de Error Estándar**

  Cuando algún comando arroja un mensaje de error en la pantalla, no lo está haciendo a través de la salida estándar, sino que lo hace a través de lo que se llama la **salida estándar de error**, la que también corresponde usualmente a la pantalla.

- **Entrada Estándar**

  Es el dispositivo del que se espera recibir los datos, usualmente corresponde al teclado del terminal. Gran parte de los comandos que requieren un archivo de entrada como argumento, leerán los datos de su entrada estándar si este no es entregado.

2. Variables de Ambiente

En Unix existe el concepto de **variables de ambiente**, que existen en los distintos ***Shells*** para Unix. Básicamente son variables como se entienden en los lenguajes formales, es decir, locaciones o direcciones de memoria donde se almacena información, en este caso información de ambiente para el ***Shell*** que se esté ejecutando. En Unix existen variables con distintas funciones, por ejemplo la variable **PATH** guarda las direcciones o caminos (de directorios) donde buscar un programa ejecutable, es así que cuando se ejecuta un programa que no se encuentra en alguno de los directorios del **PATH**, el ***Shell*** entrega un mensaje de error **Command not found.**.

Al referirse al contenido de una variable en Unix es necesario anteponer el símbolo **$** al nombre de la variable. Por ejemplo la variable **PATH** tiene como contenido **$PATH**, luego para imprimir en pantalla el contenido de la variable es necesario escribir en el terminal lo siguiente:

```
%> echo $PATH.
```

Para dar un valor a una variable existe el comando **set** que permite crear una variable (si no existe) con el nombre y valor dados como argumentos. Ejemplo:

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

Un ejemplo de variables en C-Shell

```
%> setenv PATH '~/.; ~/bin; /usr/bin; /bin'
%> echo $PATH
     ~/.; ~/bin; /usr/bin; /bin
%>
```

Otras variables típicas en Unix son:

HOME Almacena el directorio de la persona que está ejecutando el ***Shell***.

HOST Tiene la dirección (nombre) del computador en que se está conectado.

term En esta variable se define el tipo de terminal en uso. Esta variable es importante definirla para un correcto manejo del terminal, un problema típico es el no poder usar el comando **vi** en pantalla completa por tener mal definida esta variable.

user Contiene el *login* del usuario.

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 redireccionamiento.

noglob Los caracteres de globalización son interpretados como si mismos, sin ser reemplazados por los nombres de archivos que cumplan con el 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 carácter **"!"**, en esa posición, se coloca el número que tendrá en la lista de **history**.

savehist Número de comandos de la lista de **history** que se "recordarán" en la siguiente sesión.

Dependiendo el ***Shell*** que esté funcionando es cómo se definen las variables de ambientes. En Bournel Shell se definen con el comando **set**, pero en C-Shell además del comando **set** existe el comando **setenv** que por convención usa variables con nombres en mayúsculas (HOME, PATH son variables de ambiente de C-Shell que toman los valores de las variables **home** y **path** de Bournel Shell.

De ahora en adelante nos referiremos a las variables de ambiente y a su contenido como **$nombre**, por simplicidad.

3. Comandos Útiles

3.1 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.

Un caso muy utilizado es el de redefinir **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**.

3.2 man

Este comando permite obtener información del *Manual de referencia* acerca de títulos y palabras claves asociadas al texto, como ser comandos del ***shell***. Si **man** no encuentra la información solicitada, despliega en pantalla un mensaje de error.

La información se encuentra en directorios especiales, por ejemplo: /usr/man/, en donde se subdividen en catálogos y títulos por temas. Los manuales son creados con el editor **troff**, el cual posee caracteres especiales para el formato de la pantalla.

man busca la ayuda en el directorio dado por la variable de ambiente MANPATH, la cual se encuentra definida en el archivo de inicialización, por ejemplo: .tcshrc.

La sintaxis de este comando es la siguiente:

```
%> man palabra
```

Por ejemplo, si deseamos tener información referente al comando *ls*, debemos ingresar:

```
%> man ls
```

Lo cual arrojaría (extracto del man):

```
Ls(1)                   User Commands                   ls(1)


NAME
      Ls - list contents of directory


SYNOPSIS
      Ls [ -abcCdfFgilLmnopqrRstux1 ] [names]


AVAILABILITY
      SUNWcsu
DESCRIPTION
```

```
          For each directory argument, ls lists the contents of the
```

Ahora si se ingresa una *palabra* que no existe entre los títulos o palabras claves del *man*, este arroja en pantalla el siguiente error:

```
No manual entry for palabra
```

3.3 apropos

Este comando, es como un índice del **man**, o sea, como el índice de las páginas amarillas. Permite buscar comandos asociados a un string y desplegar en pantalla la ubicación dentro del manual (sección, títulos, etc.) y un comentario anexo. La información necesaria para realizar esta operación se encuentra almacenada en una base de datos creada por **catman***;* por lo cual, cuando este comando no se encuentre disponible, querrá decir que la base de datos no está levantada.

Una vez desplegada la información, se puede utilizar el comando **man** para obtener información de ellos.

Por ejemplo: Si deseo obtener los comandos que tengan relación con *dvi*, debo escribir:

```
%> apropos dvi
```

Lo cual arroja como resultado:

```
flock       flock (3b)  - apply or remove an advisory lock on an open file
madvise     madvise (3) - provide advice to VM system
xdvi        xdvi (n)    - DVI Previewer for the X Window System
```

3.4 crontab

Este comando permite dejar una tarea para que sea ejecutada por el computador a la hora, día o fecha indicada en <archivo>.

La sintaxis de este comando esta dada por:

```
%> crontab [opciones] <archivo>
```

Las opciones para este comando son:

-e [username] : Edita el <archivo> entregado a **crontab**

-l [username] : Muestra el archivo entregado a **crontab**

-r [username] : Remueve el archivo entregado a **crontab**.

Para instalar el <archivo> que se desea ejecutar, se debe hacer:

```
%>crontab <archivo>
```

El <archivo> que se le entrega a **crontab** es el que le dice el momento en que debe ejecutar el comando o script. Este comando o script también se le entrega en el <archivo>. la estructura de <archivo> es la siguiente:

Minutos	Horas	Día del Mes	Mes	Día Semana	Comando
00	00	15	*	*	users

Vemos que en MES hay un *, lo que quiere decir que "users" se ejecutara todos los 15 de cada mes del año a las 00 horas con 00 minutos. El "Día de Semana" se puede entregar como un rango, por ejemplo, 1-5, o como días específicos de la semana, i.e. 1,5,6.

4. Programación en Shell

Los archivos *Shell Script* son los archivos que son interpretados paso a paso por el Shell. Estos archivos son el equivalente a lo que son: los archivos **".BAT"** en **DOS**, aunque más importante en Unix, ya que este tipo de archivo es donde se define el ambiente de trabajo del usuario, haciéndolo más agradable o fácil de usar.

En esta sección nos referiremos mayormente a la programación de *scripts* de Bournell Shell por ser considerado el interprete Shell oficial de Unix, pero también se tratará C-Shell que tiene varias particularidades útiles (existe también TC-Shell que es un C-Shell mas un editor de línea y un sistema que completa nombres de archivos).

Es importante hacer notar que para que un archivo de Unix se pueda ejecutar debe tener permiso de ejecución. Para esto se debe usar el comando:

```
% chmod +x archivo
```

4.1 Niveles de programación en Shell

Los programas varían en complejidad. Pueden ser una simple serie de instrucciones estándar de Unix, ingresados mediante un editor, o pueden ser un grupo de programas y loops anidados. Para esto existen las siguientes estructuras de control: **if**, **case**, **for** y **while**.

4.1.1 echo

Comando de Unix que despliega el argumento en la salida estándar.

Ejemplo :

```
%> echo Hola, este es un ejemplo
Hola, este es un ejemplo
%>
```

también se puede pasar como parámetro una variable de ambiente , por ejemplo:

```
%> echo $HOST
cipres.cec.uchile.cl
%>
```

4.1.2 Parámetro

Los archivos Shell Script almacenan 10 variables, los argumentos pasados al programa (llamadas variables posicionales), estas variables incluyen como un argumento el nombre con que fue ejecutado el programa. Las variables son representadas por números de 0 (cero) a el número 9.

Veamos el siguiente archivo de ejemplo, cuyo nombre es *prueba*:

```
#!/usr/tcsh
echo $0
echo $1
echo 2
echo $3
```

Al ejecutarlo tenemos:

```
%> prueba uno dos tres cuatro
prueba
uno
2
tres
%>
```

Como se puede apreciar del ejemplo en la variable $0 se almacena el nombre con que fue llamado el programa. Es importante recordar que al no colocar el símbolo $ en la segunda línea del archivo el número 2 es considerado como parámetro del comando echo y no como la variable $2. Otras variables importantes de conocer son: **$#** es el número de argumentos recibidos y **$$** el identificador del proceso asociado al programa.

4.1.3 for

El **for** permite realizar una tarea repetitiva mientras una variable toma valores de una lista que es pasada como argumento de la instrucción.

Un ejemplo de *shell script* es el siguiente archivo de nombre *prueba*.

```
#!/usr/tcsh
for nombre in Juan Karen Pedro Luis
do
   echo $nombre
done
```

Al ejecutar: `%> prueba,` ocurre lo siguiente:

```
%> ejemplo
Juan
Karen
Pedro
Luis
%>
```

como se puede ver del ejemplo la sintaxis es:

```
for variable in lista
do
   comandos
done
```

Los comandos son ejecutados una por cada palabra de la *lista*, con la *variable* tomando un valor distinto de la *lista* por cada ciclo. Es importante recordar que en Unix al referirse al valor de una variable de ambiente se hace colocando el nombre de la variable precedido por el signo "**$**" es por eso que en el ejemplo anterior se usó `echo $nombre`, para que se mostrara en pantalla el valor de la variable **nombre**

4.1.4 while

Es un ciclo similar al for excepto que usa el estatus de salida para determinar cuando salir del ciclo.

La sintaxis del while es la siguiente:

```
while test expresión
do
   comandos
done
```

Un ejemplo de uso de **while** es (llamémoslo *mostrar*):

```
#!/usr/tcsh
While test $# != 0
Do
   Echo argumento $1
Shift
Done
```

Luego al ejecutarlo se tiene:

```
%> mostrar uno dos tres
argumento uno
argumento dos
argumento tres
%>
```

Con el comando **shift** se logra la acción de *shiftear* las variables posicionales, decrementando el número de parámetros **$#**.

4.1.5 if

La estructura de **if** es similar a la utilizada en la mayoría de los lenguajes, sirve para realizar ciertas acciones si se cumple una condición y otras acciones si no.

Existen tres formas básicas de **if**:

- **if then**

```
if test expresión
   then
   comandos
   fi
```

- **if then else**

```
if test expresión
   then
   comandos
   else
   comandos
   fi
```

- **if then else if**

```
if test expresión
   then
   comandos
   elif test expresión
   comandos
   else
   comandos
   fi
```

En las tres formas anteriores se usa el comando **test** para evaluar una expresión, pero no es necesario ya que al igual que el while usa el estatus de salida para decidir.

La tercera forma es la que se utiliza para tener if anidados.

Un ejemplo de uso de if es el siguiente:

Este ejemplo tiene como propósito el verificar la existencia de los archivos cuyos nombres se pasan como argumentos.

Un ejemplo que no utiliza el comando **test**:

```
#!/usr/tcsh
if grep CEC "$1" > /dev/null
then
   echo $1 contiene el patrón CEC
else
   echo $1 no contiene el patrón CEC
fi
```

El objetivo de este ejemplo es el ver si en un archivo existe la palabra CEC, en este caso el if pregunta por el estado de salida del comando:

```
grep CEC "$1" > /dev/null
```

que retorna 0 si no encuentra CEC en el primer parámetro .

4.1.6 case

Es un comando que permite hacer múltiples bifurcaciones, reemplazando a un grupo de if anidados. Este comando es particularmente usado cuando se hacen múltiples comparaciones de strings, su estructura es la siguiente:

```
case word in
   patrón 1)
      comandos ;;
   patrón 2)
      comandos ;;
...
   patrón n)
      comandos ;;
esac
```

Si algún patrón coincide con uno de los patrones se ejecutan las acciones asociadas a ese patrón.

4.1.7 test

Como hemos visto hasta aquí el comando **test** es muy importante para los comandos **if** y **while**, ya que sirve, entre otras cosas, para evaluar expresiones. **test** recibe argumentos y opciones que evalúa y dependiendo del tipo de opciones termina su ejecución con un 0 (es en cierta forma el falso) u otro valor .

Algunas de las opciones del comando **test** son:

-r *nombre* Verdadero si el archivo o directorio de nombre *nombre* existe y tiene permiso de lectura.

-w *nombre* Verdadero si existe y tiene permiso de escritura.

-f *nombre* Verdadero si existe y es un archivo.

-d *nombre*	Verdadero si existe y es un directorio.
-s *nombre*	Verdadero si existe y el largo es mayor que cero.
-z *s1*	Verdadero si el largo del string *s1* es cero.
-n *s1*	Verdadero si el largo del string es distinto de cero.
s1= s2	Verdadero si los string *s1* y *s2* son idénticos.
s1!= s2	Verdadero si son distintos.
s1	Verdadero si *s1* no es un string null
n1-**eq** *n2*	Verdadero si n1 y n2 son algebraicamente iguales.
n1-**gt** *n2*	Verdadero si n1 es mayor que n2.
n1-**ge** *n2*	Verdadero si n1 es mayor o igual que n2.
n1-**lt** *n2*	Verdadero si n1 es menor que n2.
n1-**le** *n2*	Verdadero si n1 es menor o igual que n2.
-a	Operador lógico AND.
-o	Operador lógico OR.
(*expr)*	Admite paréntesis para agrupar.

Es importante hacer notar que no es necesario colocar siempre la palabra test en las condiciones de while o if, se puede colocar el *bracket izquierdo* (**[**) en reemplazo de la palabra **test** y un *bracket derecho* (**]**) para balancear la instrucción.

Ejemplo :

Algunas equivalencias		
While test $#!=0	*equivalente a*	**while [$# != 0]**
if test -s $1	*equivalente a*	**if [-s $1]**
if test -d $1	*equivalente a*	**if [-d $1]**

4.1.8 Acerca de las comillas en Shell

Es importante saber que en la programación en Shell es distinta la función que usan distintos caracteres, lo que complica el aprendizaje de este *"lenguaje"*. Estos caracteres son: # * ? [ ] { } ( ) < > " ´ ' & ; $

En esta sección sólo nos referiremos a cuatro caracteres especiales:

- **Doble comillas (")** previene que sean interpretados los caracteres especiales (excepto $).
- **Comillas simples (')** anula el que cualquier caracter especial sea interpretado.
- **Acento "al revés" (`)** permite que se reemplace el string por el resultado del comando especificado por este.

A continuación se muestran ejemplos de como funciona el uso de estos caracteres:

```
%> echo este          es un ejemplo
este es un ejemplo
%>  echo " este          es un ejemplo"
este          es un ejemplo
%>  echo "$HOST"
cipres.cec.uchile.cl
%>  echo '$HOST'
$HOST
%> echo pwd
pwd
%> echo 'pwd'
/home/cipres/grupo1/dcc/alum/xjx
%>
```

5. Análisis de los archivos iniciales de Unix

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.
- **.tcshrc** : Se ejecutan los comandos que contiene cada vez que se llama al TC-Shell (al momento de conectarse y cada vez que se da el comando **tcsh**, 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 esta 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 orden.

También se pueden agregar algunas **variables de ambiente**, con lo que se puede definir el modo de trabajo dentro del TC-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 redireccionamiento.

noglob Los caracteres de globalización son interpretados como si mismos, sin ser reemplazados por los nombres de archivos que cumplan con el 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 carácter "!", en esa posición, se coloca el número que tendrá en la lista de **history**.

savehist Número de comandos de la lista de **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
```

6. Redireccionamiento de Comandos

6.1 Redireccionamiento de la salida de un comando

6.1.1 Operador >

Si un comando escribe datos a la pantalla, (que no son mensajes de error) podemos decirle que en lugar de mostrarlos en la pantalla cree un archivo y los almacene en el ; esto se realiza con el operador >.

El formato de uso de este operador es:

```
comando >arch
```

Con esta sintaxis, al ejecutarse el comando, se creara un archivo de nombre arch y en el se grabara la salida standard del comando en lugar de escribirse a la pantalla.

Por ejemplo:

```
%> ls >listado
```

6.1.2 Operador >>

Si uno quisiera guardar en un mismo archivo la salida de varios comandos consecutivos, o agregar esta al final de un archivo dado, no podríamos usar >, pues este crearía un nuevo archivo al ejecutar el comando y borraría los contenidos del anterior. Para un problema como este, existe el operador >>.

El formato de uso de >> es:

```
comando >>arch
```

>> toma la salida estandard de un comando y la agrega al final del archivo especificado, sin borrar los contenidos anteriores de este. Si arch no existe, será creado y en el se almacenara

la salida de comando, pero si existe, la salida de comando será agregada al final del archivo.

Por Ejemplo:

```
$ ls >>mi_archivo
```

6.2 Redireccionamiento de los errores de un comando

Para redireccionar la salida estandard de errores, se utilizan los mismos operadores > y >>, pero se les antepone un 2. De este modo:

```
comado 2> archivo
```

creará un archivo de nombre arch y grabará en este los mensajes de error entregados por comando.

Ahora si se ingresa el comando:

```
comando 2>> arch
```

Si arch no existe, será creado y en el se almacenarán los mensajes de error entregados por comando, pero si existe, estos mensajes serán agregados al final del archivo.

Nota:

- No se pueden incluir en un comando a la vez > y >>.
- Si se usa el operador > o >>, se dice que se esta redireccionando la salida estándar.

6.3 Redireccionamiento de la entrada de un comando

6.3.1 Operador <

Si un archivo lee datos desde el teclado, se le puede indicar que en lugar de hacerlo de este lo haga de un archivo. Esto se realiza usando el operador < cuyo formato es:

```
comando <arch
```

Lo cual indica que el comando leerá los datos desde el archivo, el cual deberá existir, sino el comando retornará un error.

Por Ejemplo: Si uno ha escrito una carta en un archivo de nombre *car* la puede enviar con:

```
%> mail usuario < car
```

Nota:

- En un mismo comando, se pueden redireccionar a la vez la salida y la entrada:

  Por Ejemplo:

```
%> cut -d: -f1,5 < /etc/passwd > usuarios
```

6.4 Conectar la salida de un comando a la entrada de otro

6.4.1 ¿Para qué?

Si uno quisiera por ejemplo hacer un listado de archivos y después verlo con un more lo puede hacer redireccionando la salida de un ls a un archivo y después haciendo un more de este, pero esto es poco práctico, pues implica tener que crear un archivo extra lo cual puede o no ser necesario o puede no ser posible. Por lo anterior, es útil saber como conectar la salida de un comando a la entrada de otro sin tener que crear archivos entre medio.

6.4.2 Operador | (pipe)

Para resolver este problema existe en UNIX lo que se denomina pipe, que es una forma de conectar la salida de un comando, a la entrada de otro de modo que ambos funcionan a la vez y no se crea ningún archivo extra.

Un pipe se realiza con el operador |, cuyo formato es:

```
comando1 | comando2
```

esto ejecutará el comando1, indicándole que entregue su salida como entrada al comando2, el cual a su vez se ejecutará con dicha entrada.

Por Ejemplo: un ejemplo práctico y muy usado es él

```
%> ls | more
```

Nota:

- En una misma línea, se pueden conectar varios comandos con |:

```
comando1 |comando2 |comando3 | ...
```

7. Manipulación de archivos

7.1 Introducción

Este módulo muestra al usuario como realizar tareas avanzadas en manipulación de archivos usando comandos que permitan manipular los contenidos de un archivo o la salida de un comando.

7.2 Búsqueda de secuencias de caracteres:

7.2.1 ¿Para qué?

Puede ser interesante saber que archivos tienen determinada secuencia de caracteres (en UNIX, los llamaremos strings), o saber las líneas de un archivo que contienen dicha secuencia.

7.2.2 Concepto

En UNIX, una secuencia de caracteres cualesquiera, con un cierto orden (uno es el primero, otro es el segundo, etc.) es denominada string.

7.2.3 Comando grep

- **Formato:** grep [-vclin] string archivo ...

- **Funcionamiento:** Este comando busca dentro de uno o más archivos un determinado string. Si no se le especifica opción alguna, grep imprimirá en la salida estándar, líneas con el siguiente formato:

- Si se le dio más de un archivo como argumento: archivo : línea

- Si se le dio sólo un archivo como argumento: línea

Donde archivo es el nombre del archivo donde se encontró la ocurrencia del string y línea es la línea donde se encontró la ocurrencia.

Se le puede especificar a grep una combinación de las siguientes opciones.

- Opción -v: Con esta opción, grep imprimirá sólo las líneas que no contengan el string dado.

- Opción -c: Esta opción indica imprimir sólo el número de líneas del archivo que debieran ser impresas en lugar de mostrarlas.

- Opción -l: Esta opción indica a grep, imprimir solamente los nombres de los archivos en que aparezca el string dado.

- Opción -i: Con esta opción, grep no diferenciará letras mayúsculas de minúsculas. O sea, considerará por ejemplo: "HOLA" igual a "hola".

- Opción -n: Indica preceder cada por su número en el archivo. De este modo la salida de grep serían líneas de la forma:

archivo :número: línea

Por Ejemplo: Si tenemos un archivo de nombre *mensaje* que contiene:

"Se comunica a los Sres. usuarios que los computadores estarán apagados de 15 a 16 hrs. hasta el próximo Sábado".

Al ejecutar: %> grep -n los mensaje

Se obtendrá la salida:

1:Se comunica a los Sres. usuarios

2:que los computadores estarán

Nota:

- grep puede ser aplicado solamente a archivos de texto.
- Para poder aplicar grep a un archivo, se debe tener permiso de lectura sobre este.

7.3 Desplegar Campos o Columnas

7.3.1 ¿Para qué?

Al tener un archivo de texto en el que las líneas son registros con varios campos, puede ser interesante ver solamente ciertos campos o columnas del archivo.

También puede ser útil desplegar un archivo en varias columnas, para, por ejemplo crear una tabla mezclando las líneas de varios archivos.

7.3.2 Comando cut

- **Formato:** Si se especifican los campos por un largo fijo: cut -clista archivo ...

  Si se especifican por un caracter separador: cut -flista [-dsep] archivo ...

- **Funcionamiento:** Este comando imprime sólo ciertas columnas de un archivo. Si uno considera al archivo como una tabla de campos, cut permite seleccionar campos de la tabla. Los campos a mostrar se pueden especificar:

Con largo fijo: o sea, el largo del campo es un número fijo de caracteres situado en una posición fija de la línea.

Con un caracter separador de campos: en este caso, el largo de los campos puede variar de línea a línea y los campos en lugar de ser diferenciados por su posición en la línea, lo son por este caracter separador de campos.

Este comando entrega su salida a la salida estándar.

- **Especificando campos por largo fijo:** Se utiliza la opción -c, junto a la cual se especifican el rango de caracteres a mostrar por línea. El rango se especifica dando la posición en la línea del primer y del último caracter del campo, separados por un signo "-". Para especificar varias columnas, se especifican junto a -c, los rangos separados por comas (",").

Por Ejemplo: Teniendo el archivo *archivo1* que contenga:

```
"****CI****:*****Nombre
******:****Direccion******
13410289-8:Juan Perez: Pasaje 3, Casa Z
10323024-3:Francisco Villa: Antonio Varas 345. Dpto 25
06763471-9:Rebeca Galindo: Arturo Prat 6514
```

Al ejecutar: %> cut -c1-10 archivo1

Se obtendrá como salida:

```
****CI****
13410289-8
10323024-3
06763471-9
```

- **Especificando campos por caracter separador:** Se utiliza la opción -f, especificando junto a ella el numero del campo a mostrar. Para especificar más de un campo, se señalan junto a la opción -f separados por comas. La opción -d sirve para especificar el caracter separador, el cual se escribe junto a dicha opción (sep en el formato). Si no se especifica el separador cut supondrá que el caracter separador es un tabulador (tab).

  Por Ejemplo: Con el mismo archivo del ejemplo anterior, al ejecutar:

  %> cut -d: -f1,3 archivo1

  se obtiene la salida:

```
****CI****:****Direccion******
13410289-8:Pasaje 3, Casa Z
10323024-3:Antonio Varas 345. Dpto 25
06763471-9:Arturo Prat 6514
```

7.3.3 Comando paste

- **Formato:** paste [-dlista] archivo1 archivo2 ...

- **Funcionamiento:** Este comando funciona de dos maneras:

- La primera consiste en tomar líneas de varios archivos y escribirlas una a continuación de la otra, separándolas por un caracter separador; si se piensa en las líneas de los archivos como en registros, este comando permite crear tablas que contengan todos estos registros (uno al lado del otro).

- El segundo modo de funcionamiento de paste implica tomar, en lugar de líneas de distintos archivos, líneas consecutivas de un mismo archivo y pegarlas una al lado de la otra usando el caracter separador que se le indique. Si fue especificada una lista de

separadores, las líneas que serán pegadas estarán separadas por caracteres de dicha lista, si esta no fue especificada, se usará como separador el caracter tab.

- **Pegar archivos:** Si no se especifica la opción -d, paste imprimirá a la salida estándar las líneas de los archivos especificados entregando en su primera línea de salida la primera línea de cada archivo una al lado de la otra en el orden en que fueron especificadas y separándolas por tabs, luego la segunda de cada uno, luego la tercera ... y así hasta la ultima línea del archivo más largo.

- **Pegar archivos con la opción -dlista:** Usando la opción -d, paste pegará los archivos del mismo modo pero utilizando como separadores el o los caracteres especificados en la lista de separadores lista.

La lista de separadores consiste en los caracteres que se usarán como separadores escritos uno al lado del otro.

Por Ejemplo: -d%#

IMPORTANTE

- Los caracteres que tengan algún significado especial para el shell (en particular el caracter espacio) deberán ser puestos entre comillas ("").
- Los caracteres "retorno de carro", espacio, tab, backslash ("\") y el string vacío (el cual significa no separar las líneas), se escribirán en la lista como sigue:

\n retorno de carro.

\t tabulador

\\ backslash ("\")

\0 string vacío.

Las líneas del primer archivo estarán separados de las del segundo por el primer caracter de la lista, las del segundo lo estarán de las del tercero por el segundo caracter de la lista y así sucesivamente.

La lista de separadores se ocupará en forma circular, o sea si se llega al último caracter de esta y aun quedan más archivos, paste volverá al comienzo de la lista de separadores.

Por Ejemplo: Si tenemos un archivo de nombre "num", el cual contiene de a uno por

línea los números: {10,78,69,12,13,14,2,7}; y otro archivo de nombre colores que contiene: {rojo, azul, verde, café, negro, blanco, calipso}. Al ejecutar: %> paste -d# num colores, se obtiene la salida :

```
10#rojo
78#azul
69#verde
12#cafe
13#negro
14#blanco
2#calipso
7#
```

Pegar líneas consecutivas: Usando la opción -s se le indica a paste que, en lugar de leer primero la primera línea de cada archivo, luego la segunda, etc., lea primero todas las líneas del primer archivo, luego todas las líneas del segundo y así sucesivamente.

Por Ejemplo: Con el mismo archivo num del ejemplo anterior, al ejecutar: %> paste -s -d"#\n" num, se obtiene la salida:

```
10#78
69#12
13#14
2#7
```

Con el archivo colores del ejemplo anterior: %> paste -s -d#= colores se obtiene la salida:

```
rojo#azul=verde#cafe=negro#blanco=calipso
```

Nota:

- La entrada estándar puede ser utilizada en lugar de un archivo, colocando un - en lugar de un nombre de archivo.

7.4 Localizar Archivos

7.4.1 ¿Para qué?

Suele suceder que uno olvide o simplemente no sepa donde esta un archivo de cierto nombre, o que otro usuario haya dejado un archivo en el directorio de uno. En estos y muchos otros casos similares, es útil poder encontrar estos archivos.

7.4.2 Comando find

- **Formato:** find directorio opciones

- **Funcionamiento:** El comando find busca a partir de un directorio dado (y revisando luego todos sus subdirectorios) archivos o directorios que cumplan ciertas propiedades, las cuales se indican por medio de ciertas opciones que especifican el tipo de búsqueda a realizar.

- **Opción -print:** Esta opción indica a find que imprima a la salida estandard sus resultados. En caso de que esta opción este ausente, find no mostrará los resultados de su búsqueda.

7.4.3 Buscar archivos por nombre

find directorio -name nom. Usando la opción -name, find buscará en el directorio directorio de nombre nom.

7.4.4 Buscar archivos por dueño

find directorio -user usuario. Con la opción -user, decimos a find que busque en directorio los archivos cuyo dueño sea usuario.

7.4.5 Buscar archivos por permisos

find directorio -perm num. Utilizando la opción -perm se le indica a find que busque archivos que tengan determinados permisos, los cuales se especifican en num, el cual es el permiso escrito en el modo absoluto de chmod.

7.4.6 Buscar archivos por fecha de acceso

find directorio -atime n. La opción -atime indica buscar los archivos que hayan sido accesados hace un número n de días. Los archivos accesados el día actual, find considera que fueron accesados hace 0 días. El que un archivo haya sido accesado, quiere decir que ha sido leido o escrito. Al buscar los archivos, find mismo cambia las fechas de acceso de los directorios.

7.4.7 Buscar archivos por fecha de modificación

find directorio -mtime n. Con la opción -mtime, find buscará los archivos que hayan sido modificados hace un número n de días. Si un archivo ha sido modificado el día actual, find considerará que fue modificado hace 0 días. El que un archivo haya sido modificado, quiere decir que ha sido escrito . (Obs.: al cambiar la fecha de modificación, cambia la fecha de acceso también).

Nota:

- En una misma línea, se pueden combinar varias opciones, de modo de combinar varios tipos de búsqueda. Por Ejemplo: $ find / -user usuario1 -perm 777 -print

7.5 Realizar Conversiones en Archivos

7.5.1 ¿Para qué?

Borrar ciertos caracteres de un archivo, o pasar todas las minúsculas a mayúsculas son típicos problemas para el usuario UNIX. Para este tipo de labores, existen los comandos tr y dd.

7.5.2 Comando tr

- **Formato:** tr [-d] string1 [string2]

- **Funcionamiento:** Este comando copia la entrada estándar a la salida estándar reemplazando o borrando según le sea indicado, los caracteres de dicha entrada, que estén dentro de la palabra string1. El uso de este comando suele ser dentro de pipes o redireccionando su entrada y/o salida estándar.

- **Reemplazar Caracteres:** tr copia la entrada estándar a la salida estándar reemplazando los caracteres que estén en la palabra string1 por los que estén en la misma posición en la palabra string2.

  Por Ejemplo: %> date entrega la salida:

```
Wed Jan  5 08:59:55 CDT 1994
```

  entonces:

```
%> date |tr ":0" "/#"
```

  entregará:

```
Wed Jan  5 #8/59/55 CDT 1994
```

- **Borrando caracteres con tr:** Con la opción -d, tr requiere sólo una palabra como argumento y copiará a la salida estandard todo caracter presente en la entrada estandard a menos que este esté en dicha palabra.

  Por Ejemplo: %> date, entrega la misma salida del ejemplo anterior:

  entonces:

```
%> date |tr  -d ": "
```

  entregará:

```
WedJan5085955CDT1994
```

Nota:

- En las palabras, se pueden especificar rangos de caracteres con la primera letra del rango seguida de un guión y luego, de la última palabra del rango, todo dentro de paréntesis cuadrados ([]). En algunos shells, los paréntesis cuadrados, deberán ponerse ente comillas.

Por Ejemplo: [a-z] equivale al rango de letras desde la a hasta la z y se puede usar en: $ tr "[a-z]" "[A-Z]"

7.5.3 Comando dd

El comando dd copia el archivo de entrada al archivo de salida realizando una transformación según sea especificada. Luego de finalizar la operación, dd entrega un reporte de la actividad realizada. El uso de este comando suele ser dentro de pipes.

- **Formato:** dd [opción=valor]

- **Opción if:** Esta opción permite especificar el archivo de entrada. En caso de no estar presente, dd supone que el archivo de entrada será la entrada estándar.

- **Opción of:** Esta opción permite especificar el archivo de salida. En caso de no estar presente, dd supone que el archivo de salida será la salida estándar.

- **Opción conv:** La opción conv debe estar presente pues su valor indica el tipo de conversión que se llevará a cabo. Esta opción puede tomar los valores:

  ucase : significa transformar minúsculas en mayúsculas.

  lcase : significa transformar mayúsculas en minúsculas.

  Por Ejemplo: Si el comando %> pwd entrega:

```
/home/acct/alumno1
```

  entonces:

```
%> pwd | dd conv=ucase
```

  entregaría:

```
/HOME/ACCT/ALUMNO1
0+1 records in
0+1 records out
```

Nota:

- El archivo de entrada debe ser distinto del archivo de salida.

7.6 Ordenar Archivos

7.6.1 ¿Para qué?

El usuario, puede requerir ordenar las líneas de uno o más archivos en orden alfabético. Para esto existe el comando sort.

7.6.2 Comando sort

- **Formato:** sort [-bdfnr] [-tx] [pos_campo] [-cmu] [-oarchivo] archivo ...

- **Funcionamiento:** Este comando toma todas las líneas de el (o los archivos) de entrada y las escribe a la salida estándar con cierto criterio en el tipo de orden que se le especifique.

- Con la opción -o se le puede especificar a sort que escriba sus resultados en un archivo en lugar de a la salida estándar. Por omisión, ordenara dichas líneas de menor a mayor. Para comparar, sort compara de izquierda a derecha caracter a caracter de la línea o campo por el cual se ordena.

Los criterios de escritura corresponden a las opciones:

- -c: Indica a sort que debe revisar si el archivo de entrada no esta ya ordenado según el orden especificado y solamente usara las líneas de este archivo para susalida si esta desordenado. Esta opción funciona solamente si se especifica un único archivo de entrada.

- -u: Con esta opción, si sort encuentra mas de una línea igual o si se especificó comparación por campo y hay más de una línea con el campo igual, entonces escribira a la salida solo una de tales líneas.

El tipo de orden es modificado por las opciones:

- -b: Significa no considerar para el orden los espacios al inicio de los campos.

- -d: Esto indica que un orden tipo diccionario, o sea que sólo importan las letras, los números y los espacios para la comparación.

- -f: Considera todas las letras como si fueran mayúsculas.

- -n: Compara en forma numérica, o sea reconoce el punto decimal y el signo en los números.

- -r: Revierte el sentido del orden, esto es en vez de ordenar de menor a mayor, ordenara de mayor a menor.

Para ordenar por línea completa: Si a sort no se le especifican las opciones -tx ni +pos1 ni -pos2, ordenará comparando las líneas completas.

Por Ejemplo: Si la salida de:

```
%> who
```

fuera:

```
emercade console Jan  4 10:36
scastro     ttya     Jan  5 08:39
emercade ttyp0   Jan  4 10:36  (:0.0)
emercade ttyp1   Jan  4 10:36  (trauco.dcc.uchil)
emercade ttyp2   Jan  4 11:55  (trauco.dcc.uchil)
wcontrer  ttyp8   Jan  4 09:56  (machi.med.uchile)
erodrigu   ttyp9   Jan  3 17:28  (limari:0.0)
```

Entonces ejecutando un:

```
%> who | sort -u -dfr
```

obtendré como salida:

```
wcontrer  ttyp8     Jan  4 09:56   machi.med.uchile)
scastro     ttya       Jan  5 08:39
erodrigu   ttyp9     Jan  3 17:28   (limari:0.0)
emercade ttyp2     Jan  4 11:55  (trauco.dcc.uchil)
emercade ttyp1     Jan  4 10:36  (trauco.dcc.uchil)
emercade ttyp0     Jan  4 10:36   (:0.0)
emercade console Jan  4 10:36
```

- Para ordenar por un campo: Al igual que en cut, podemos considerar las líneas como registros con campos, luego podemos escoger uno o varios de estos campos para ordenar las líneas de acuerdo a su contenido. El campo por el que se va a ordenar se especifica con un caracter separador de campos y una posición en la línea, la cual es indicada por *pos_campo*. En la línea, las posiciones de los campos parten de 0 (0 para el primero, 1 para el segundo, etc.).

- El caracter separador se especifica usando opción -t, la cual va acompañada del caracter separador de campos (x en el formato). Si no se especifica el caracter separador de campos, sort supondrá que los campos están separados por espacios o tabuladores.

- *pos_campo* se puede dar en dos formatos :

  +pos1: De esta forma, se indica a sort que debe ordenar usando los campos de la línea a partir del que este en la posición pos1.

  +pos1 -pos2: En esta forma, sort ordenará usando los campos de la línea desde el

indicado por pos1 hasta el anterior a pos2.

Por Ejemplo: Si tengo una archivo de nombre números que contiene:

```
1001 20 0 1
8888 250 0 1
6010 6969 0 1
1002 20 0 0
0 0 -2 -2
```

Al hacer: %> sort -r -n +1 números, se obtendrá:

```
6010 6969 0 1
8888 250 0 1
1002 20 0 0
1001 20 0 1
0 0 -2 -2
```

7.7 Mostrar el Comienzo o Final de un Archivo

7.7.1 ¿Para qué?

Si se sabe que cierto archivo es modificado solamente al inicio o solamente al final de este, es útil ver solo dicha parte de este, vale decir un cierto numero de líneas desde el principio del archivo o antes del final de este. Para esto existen los comandos head y tail.

7.7.2 Comando head

- **Formato:** head -n archivo ...

head muestra las primeras n líneas del archivo especificado. Si es omitido, muestra las primeras 10.

Por Ejemplo: Usando el archivo números del ejemplo anterior:

```
%> head -2 números
```

Entrega la salida:

```
1001 20 0 1
8888 250 0 1
```

7.7.3 Comando tail

- **Formato:** *tail [-|+[num] lcb] [-f] archivo*

- **Funcionamiento:** *tail* copia un archivo a la salida estándar a partir de un lugar dado. Este comando puede copiar el archivo a partir de cierta distancia inicio o del final del archivo. La distancia esta dada por num, y puede estar dada en caracteres ,líneas o bloques de 512 caracteres: Si se especifica num, debe estar precedido de un + o un -.

Opciones:

- **Opción +:** Si se especifica +, *tail* mostrará el archivo a partir de una distancia num del inicio del archivo.

- **Opción -:** Si se especifica -, *tail* mostrará el archivo desde una distancia num del final del archivo.

- **Opción l:** Si se agrega l a num, indica que la distancia esta en líneas.

- **Opción c:** Si se agrega c a num, indica que la distancia esta en caracteres.

- **Opción -b:** Con esta opción, se le indica a *tail* que los pedazos en lugar de ser de num líneas, serán de num bloques de 512 caracteres.

- **Opción -f:** Esta opción permite "ver" como crecen archivos que aumentan constantemente de tamaño. Especificando -f , tail mostrará la parte del archivo que le haya sido especificada, y luego quedará esperando por si alguien agrega datos al final del archivo, en cuyo caso los irá mostrando a continuación del pedazo ya mostrado.

Por Ejemplo: Usando nuevamente el archivo números del último ejemplo de sort:

```
%> tail -2c números
```

Mostrará la salida:

```
1002 20 0 0
0 0 -2 -2
```

Nota:

- Si no se especifica num, *tail* supone que la distancia es 10.
- Si no se especifica ni l ni c, *tail* supone que la distancia esta dada en líneas.

7.8 Comprimir / Descomprimir Archivos

7.8.1 ¿Para qué?

En todo computador, el espacio es limitado, luego es útil poder comprimir archivos de gran tamaño para ahorrar espacio. La idea de comprimir un archivo, es la misma de desinflar un globo, si uno le vuelve a echar aire al globo recupera el globo como originalmente fue, al igual, al descomprimir un archivo este vuelve a ser el archivo original.

7.8.2 Comprimir archivos

Comando compress

- **Formato:** compress archivo ...

  compress comprime el archivo indicado y creando un archivo con el mismo nombre que el original pero agregando la extensión ".Z".

Comando pack

- **Formato:** pack archivo ...

  al igual que compress, pack comprime el archivo indicado, pero genera un archivo de extensión ".z" y al terminar de comprimir, entrega un mensaje avisando si comprimió correctamente el archivo y el porcentaje de compresión logrado.

Comando zip y gzip

- **Formato:** zip archivo <lista>

gzip archivo <lista>

Comprimen la <lista> indicada, generando un archivo de extensión ".zip" y ".GZ" respectivamente.

Nota:

- Tanto pack como compress, si logran comprimir correctamente el archivo (no hay errores de ningún tipo), borrarán el original.
- Algunos archivos se comprimen más con compress y otros con pack.
- Los archivos comprimidos que generan pack y compress, conservarán los atributos de los originales.
- Actualmente son mayormente utilizados los compresores zip y gzip

7.8.3 Descomprimir archivos

Comando uncompress

- **Formato:** uncompress archivo ...

Este comando sirve para descomprimir los archivos comprimidos con compress (aquellos con extensión ".Z"). El argumento archivo, puede ser el nombre del original o el nombre del archivo comprimido, o sea con la extensión .Z. uncompress regenerará el archivo original con los atributos que tenía el comprimido y luego borrará este.

Comando unpack

- **Formato:** unpack archivo ...

Este comando sirve para descomprimir los archivos comprimidos con pack (los con extensión ".z"). El argumento archivo, puede ser el nombre del original o el nombre del archivo comprimido, o sea con la extensión ".z". Al igual que uncompress, unpack regenerara el original con los atributos del comprimido y después borrará este último, pero además escribe un mensaje en pantalla avisando que descomprimió correctamente el archivo.

Comando unzip y gunzip

- **Formato:** unzip archivo ...

  gzip archivo...

Estos comandos sirven para descomprimir los archivos comprimidos con zip y gzip respectivamente (los con extensión ".zip" y “.GZ”). El argumento archivo, puede ser el nombre del original o el nombre del archivo comprimido, o sea con la extensión ".zip" (“.GZ”). Al igual que uncompress y unpack regenerará el original con los atributos del comprimido.

7.8.4 Mostrar el contenido de archivos comprimidos

Comando zcat

- **Formato:** zcat archivo ...

zcat descomprime archivos comprimidos con compress, pero en lugar de regenerar el archivo original, escribe sus contenidos a la salida estandard. archivo puede ser indicado del mismo modo que para uncompress, o sea puede o no escribirse con la extensión ".Z".

Comando pcat

- **Formato:** pcat archivo ...

pcat descomprime archivos comprimidos con pack pero, del mismo modo que zcat, en lugar de regenerar el archivo original, escribe sus contenidos a la salida estándar. Archivo puede o no escribirse con la extensión ".z".

Nota:

- Para ver el contenido de archivos comprimidos con **zip** se utiliza la opción **–t.**

```
%>zip –t <archivo>
```

7.8.5 Duplicar la Salida de un Comando

¿Para qué?

Si un comando entrega datos a la salida estándar, puede suceder que el usuario a un mismo tiempo quiera ver esos datos o usarlos en un pipe y a la vez grabarlos en un archivo. Para esto se tiene el comando tee.

Comando tee

- **Formato:** tee [-a] arch

- **Funcionamiento:** tee crea el archivo arch y luego recibe datos en la entrada estándar, los cuales escribe a la vez en arch y en la salida estándar.

  El uso de tee suele ser dentro de pipes, de modo de poder usar la salida de un comando como entrada para otro y al mismo tiempo grabarla en un archivo.

- **Opción -a:** Con esta opción si el archivo archivo existe, en lugar de crearlo de nuevo, con lo que borraría sus contenidos anteriores, escribirá después del final del

archivo.

Por Ejemplo: %> ls -al | tee listado, este comando entregará en pantalla la salida normal de un ls, pero además la grabará en un archivo de nombre listado.

8. X-WINDOW

8.1 Configuración inicial

8.1.1 Archivo .xinitrc

El archivo .xinitrc es un shell script que es llamado al iniciarse una sesión de X-Window. Este archivo tiene dos requisitos:

1. Todas sus líneas (excepto la última) deben terminar con &.

2. En la ultima línea (sin &) debe ejecutarse un Windows Manager (FVWM, OLWM, etc.)

8.1.1.1 Fondos

Hay varias maneras de poner fondos:

- xv -root -quit archivo-de-imagen
- xsetroot [opciones]

  opciones:

  - -solid *color*: en que *color* es un string con el nombre de un color
    - black
    - red
    - yellow
    - gray
    - la lista completa de colores se puede obtener con el comando *showrgb.*
  - -gray
  - -grey
- xphoon : fondo lunar.
- xhearth : fondo terrestre.

8.1.1.2 Otras Configuraciones Iniciales

Además desde este archivo se pueden ejecutar todas las aplicaciones que se desea se ejecuten al iniciar una nueva sesión de X-Window.

8.1.2 Configuración de las Aplicaciones

Para setear la presentación de las aplicaciones X-Window son principalmente iguales en todas las aplicaciones:

8.1.2.1 Elección de colores

-fg *color* : setea el color del texto como "*color*".

-bg *color* : setea el color del fondo como "*color*".

-bd *color* : setea el color del borde de la ventana como "*color*".

-name *name* : setea el nombre de la ventana creada como "*name*", bajo este nombre será conocida la aplicación.

-title *title* : setea el titulo (en la barra de titulo) como "*title*".

-rv : video inverso (intercambia los colores de fg y bg)

-display : especifica el servidor X al que se debe conectar (nombre de la maquina en que aparecerá la aplicación).

-iconic : genera una ventana iconificada.

-bw *#n* : setea el grosor del borde (en "*#n*" pixeles)

8.1.2.2 Geometría de las Aplicaciones

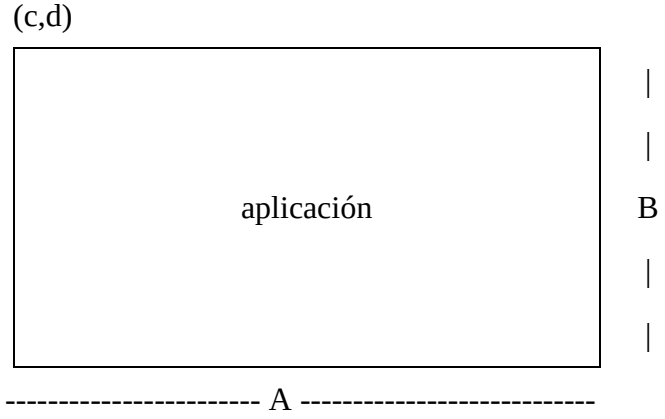
Las aplicaciones para X-Window tienen una opción -geometry con la que se le indica el tamaño y la posición de la ventana. Su sintaxis es la siguiente:

USO:

nombre-aplicación [opciones-varias] -geometry AxB+C+D

Funcionamiento:

Que significa que la ventana tendrá un ancho A, un alto B y su esquina superior izquierda estará en la posición (C,D).



8.2 Windows Managers (.fvwmrc)

El Sistema X-Window necesita de un administrador de ventanas (Windows Managers)para poder manejar estas. existen varios (twm, olwm, olvwm, fvwm, fvvwm95, etc.) siendo el más utilizado fvwm y ahora, recién salido del horno, fvwm95 (fvwm con interfaz Windows 95). Estos administradores de ventanas utilizan archivos de configuración. Nos dedicaremos a estudiar fvwm, cuyo archivo de configuración es .fvwmrc.

8.2.1 algunas configuraciones generales

- AutoRaise n : selecciona la ventana sobre la que esta el puntero después de n milisegundos.
- ClickToFocus : espera el click en el mouse para seleccionar la ventana.
- IconBox n1 n2 n3 n4 : Los iconos quedan ordenados en el área definida por las coordenadas n1 n2 n3 n4.
- StickyIcons : Los iconos quedan pegados (aunque te cambies de ventana, los iconos se mueven contigo.
- OpaqueMove n : Hace que las ventanas que ocupen menos del n% de la pantalla, no se pongan opacas al moverse.
- EdgeScroll n n : se mueve n% la pantalla al deslizarse por el escritorio virtual.
- ModulePath : es el path en que se encuentran los módulos a ejecutar durante una sesión del Fvwm. en las maquinas del CEC se debe poner:

ModulePath
/local/homes/dsp/nation/modules:/usr/lib/X11/fvwm:/usr/local/X11R6/lib/fvwm/

Luego para ejecutar un modulo se debe escribir Module module-name.

- NoTitle nombre-aplicación : Hace que la ventana de la aplicación nombre-aplicación, aparezca sin barra de titulo.
- NoBorder nombre-aplicación : Hace que la ventana de la aplicación nombre-aplicación, aparezca sin bordes.
- Sticky nombre-aplicación : hace que la ventana de la aplicación nombre-aplicación, se quede pegada (aunque uno se desplace por el escritorio virtual).
- StaysOnTop nombre-aplicación : hace que la ventana de la aplicación nombre-aplicación, se mantenga siempre sobre las demás aplicaciones (las tapa).
- WindowListSkip nombre-aplicación: La lista de aplicaciones de X-Window ignora la aplicación.

8.2.1.1 Iconos e Imágenes

Para setear la ubicación de los iconos e imágenes:

pixmapPath /usr/local/include/X11/pixmaps/

iconPath /usr/local/include/X11/pixmaps/

o directorios a tu elección donde tengas tus propias imágenes.

y luego con la línea:

icon “aplicación” *icono*

se le asocia el icono de nombre “*icono”* a la aplicación. “*icono*” debe estar en los path definidos.

8.2.2 Definiciones de fonts en uso:

Font *font* : Font de los popup

WindowFont *font* : Font para la barra de titulo en las ventanas .

PagerFont *font* : Font para el FvwmPager (nombre en Ventanillas).

IconFont *font* : Font para nombrar los iconos.

La sintaxis para la variable font es la siguiente:

```
-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

En que cada asterisco (*) se reemplaza por un valor según la posición del asterisco. (xfontsel es muy útil para elegir buenos fonts).

8.2.3 configuración en general de las ventanas.

8.2.3.1 Definición de colores:

Colores de las ventanas no seleccionadas:

StdForeColor *color* : color de las letras del Titulo.

StdBackColor *color* : color de los bordes y barra de titulo.

Colores de las ventanas seleccionadas:

HiForeColor *color* : color de las letras del Titulo.

HiBackColor *color* : color de los bordes y barra de titulo.

Colores del Escritorio Virtual (FvwmPager)

PagerBackColor *color*: color del fondo.

PagerForeColor *color* : color divisiones y pantalla actual.

Colores de aplicaciones pegadas (Sticky).

StickyForeColor *color* : color de las letras del Titulo.

StickyBackColor *color* : color del bordes y barra de titulo.

Colores de los popup.

MenuBackColor *color* : color del fondo.

MenuForeColor *color* : color de las letras.

8.2.3.2 Botónes en la Barra de Titulo.

Se pueden definir botones en la barra de titulo, y que tengan una acción asociada sobre la ventana (maximizar, minimizar, mover, cambiar tamaño, etc.). Un ejemplo de la sintaxis es el siguiente:

#	Botón	Contexto	Modificador	Función
Mouse	0	1	A	Function "windows_ops_func"
Mouse	0	2	A	Function "Resize-or-Raise"
Mouse	0	4	A	Iconify

En que :

botón 0 : significa que cualquier botón del mouse hará la acción.

Contexto 1,2,4 : (0-9) son los botones en la barra de titulo.

Modificador A : Significa que no hay que apretar ninguna tecla extra.

Y las funciones Windows_ops_func, Resize-or-Raise e Iconify están previamente definidas.

8.2.4 Configuración del Escritorio Virtual (Desktop) y su Pagina (Pager).

DeskTopSize *NxM* : Fija un escritorio de NxM ventanas.

DeskTopScale *n* : escala del escritorio virtual con respecto a la pantalla.

Pager *n m* : pone el FvwmPager en la posición (n,m).

8.2.5 Configuración de los Popup

Al presionar un botón del mouse mientras el cursor esta sobre el fondo de X-Window, aparecen los menús colgantes (popup) cada botón tiene un popup asociado.

La sintaxis es la misma que los botones en la barra de titulo de la parte configuración de ventanas virtuales. Un ejemplo seria el siguiente:

#	Botón	Contexto	Modificador	Función
Mouse	1	R	A	Popup “utilities”
Mouse	2	R	A	Popup “Windows Ops”
Mouse	3	R	A	WindowList

En que :

botón 1,2,3 : Indica el botón del mouse que desplegara el popup.

Contexto R : Mientras el cursor del mouse este en el fondo (RootWindows).

Modificador A: Significa que no hay que apretar ninguna tecla extra.

Y los popup Utilities, Windows Ops, y la función WindowList están previamente definidos.

Los popup se definen de la siguiente forma:

popup “Nombre_Popup”

Title “Titulo_del_Popup”

“Función” “descriptor”

endpopup

En que:

“descriptor” : debe ir entre comillas “ y será lo que aparece en el menú.

“Función” : puede venir definida en fvwm o estar predefinida por el usuario.

8.2.6 Definición de Funciones

En caso de estar predefinida por el usuario, la sintaxis sería:

Function “descriptor” nombre_función

Algunas de las funciones predefinidas por FVWM son:

1. popup “descriptor” popupname : llama al popup de nombre popupname.
2. Nop “descriptor” : no hace nada, si el descriptor es vacío pone una línea horizontal (separador) en el menú.
3. Title “title_popup” : define el titulo del popup como title_popup.
4. Refresh : Refresca la pantalla.
5. Function “descriptor” functionname : llama a la función functionname.
6. Raise : trae al frente una ventana.
7. Exec “descriptor” exec comando : ejecuta comando como si fuera ejecutado desde el shell.
8. Lower : lleva hacia el “fondo” una ventana (inverso de Raise).
9. Iconify : (De)iconifica una ventana.
10. Sticky : (Des)pega una ventana.
11. Maximize : agranda una ventana según algunos parámetros.
12. Destroy : hace un “kill -9 -1” de una ventana (mata la aplicación completa asociada a esa ventana).
13. Delete : borra una (y solo una) ventana.
14. Quit : sale de fvwm (y normalmente de X-Window).
15. Restart “descriptor” WinMan : ejecuta el WindowsManager de nombre WinMan.
16. Module “descriptor” ModuleName : ejecuta el Modulo de nombre ModuleName.

Resumen

En resumen, se puede definir una serie de POPUP, crear BOTONES (en la barra de titulo de una ventana, usar los botones del mouse, usar el teclado, o usar combinaciones) de modo que al accionar un botón se despliegue un popup, o simplemente se ejecute una acción.

Para crear los distintos botones, aquí va un resumen:

La línea de “comando” es la siguiente:

Mouse BOTÓN CONTEXTO MODIFICADOR FUNCIÓN

Y los parámetros BOTÓN, CONTEXTO, MODIFICADOR y FUNCIÓN significan:

BOTÓN : botón del mouse a utilizar.

CONTEXTO : Lugar en que debe estar el cursor.

MODIFICADOR : indica que tecla se debe presionar junto con el botón.

FUNCIÓN : Función que se ejecutara. Puede tomar valores predefinidos o incluidos en FVWM.

BOTÓN, CONTEXTO y MODIFICADOR pueden tomar los siguientes valores:

BOTÓN	CONTEXTO	MODIFICADOR
1 Primer Botón del Mouse	R Root Windows (fondo)	N Sin Modificador
2 Segundo Botón del Mouse	W Ventana de Aplicación	C Control
3 Tercer Botón del Mouse	T Barra de Título de una Ventana	S Shift
0 (Cualquier Botón Sirve)	S Borde de una ventana	M Meta
--	F Esquina de los bordes	A Cualquiera
--	I Sobre un ícono	o una combinación de ellas.
--	0-9 Botones en la Barra de	--

	Título	
--	o una combinación de las letras	--
--	A es para Todos los Contextos excepto los Botones en Barra de Título	--

8.2.7 Configuración del GoodStuff

Cuando se esta trabajando con una interfaz gráfica (X-Window por ej.) es útil tener una barra de botones, Fvwm provee una llamada GoodStuff. Es un sector de la pantalla dividida en botones, cada botón tiene asociada una aplicación, por lo que basta hacer click para que esta se active.

Para activar el GoodStuff se usa la línea:

Module GoodStuff

Luego se debe configurar.

La mayoría de los comandos de FVWM que hemos visto son idénticos para el GoodStuff (NoTitle, Sticky, StaysOnTop, WindowListSkip,etc). Pero además existen algunas configuraciones extras o que pueden utilizarse de más de una forma:

Style “GoodStuff” NoTitle,NoHandles,BorderWidth 0

GoodStuffFore *color* : Color Fondo.

GoodStuffBack *color* : Color Divisiones.

GoodStuffFont *font* : tipo de letras del descriptor (formato de xfontsel).

GoodStuffGeometry *AxB+C+D* : define la geometría de del GoodStuff.

GoodStuffColumns *n* : Genera un GoodStuff de n columnas.

GoodStuffRows *m* : Genera un GoodStuff con m filas.

8.2.7.1 Configuración de los elementos (botones) del GoodStuff.

El formato es el siguiente:

*GoodStuff descriptor Imagen FUNCIÓN

Imagen : es un archivo de imagen (generalmente xpm)

FUNCIÓN : es una función previamente definida (incluyendo exec para ejecutar cosas como si fuera el shell).

8.3 Aplicaciones Útiles Básicas PARA X-WINDOW

Cuando uno esta en una estación de trabajo, hay una serie de aplicaciones útiles básicas, que permiten hacer más productivo y agradable el uso de esta. Algunas de estas aplicaciones son:

1. Xload
2. Reloj
3. Correos (xbiff)
4. Ojos (xeyes)

Para todas ellas son validas las opciones mencionadas en la parte Configuración de las Aplicaciones.

8.3.1 XLOAD

La aplicación *xload* permite mostrar un histograma de la carga de la estación , que se actualiza periódicamente.

USO:

% xload [opciones]

Opciones:

-hl [color] : Esta opción especifica el color de las líneas de escala

-label [nombre] : Titulo para la ventana del xload.

-scale [Numero Entero] : Especifica la escala del histograma.

-update [segundos] : Especifica cada cuantos segundo se actualizara el histograma, los valores deben estar entre 1 y 5 segundos.

8.3.2 XCLOCK (Reloj)

La aplicación *xclock muestra un reloj*.

USO:

% xclock [opciones]

Opciones:

-analog : selecciona un reloj análogo.

-digital : Selecciona un reloj digital

-chime : El reloj suena cada media hora.

-hd [color] : Especifica el color de los punteros del reloj(análogo)

-hl [color] : Especifica el color de los bordes de los punteros del reloj.

-update [segundos] : Especifica cada cuantos segundos se actualiza la ventana del Reloj.

8.3.3 XBIFF(Correo)

El programa xbiff muestra una pequeña imagen de un buzón, el que muestra una bandera cuando ha llegado un mail

USO:

% xbiff <opciones>

Opciones:

-update [segundos] : Especifica la frecuencia en segundos con que xbiff actualizará la ventana.

-file [archivo] : Esta opción especifica el archivo que se quiere monitorear.

-display [display]: Esta opción especifica el servidor a contactar.

8.3.4 XEYES (Ojos)

Abre una ventana con unos ojos que siguen el cursor.

USO:

% xeyes <opciones>

Opciones:

-fg [color] : Selecciona un color para la pupila de los ojos.

-bg [color] : Selecciona un color para el fondo de la ventana de los ojos.

-outline [color] : Selecciona un color para la línea del borde de los globos oculares.

-center [color] : Selecciona un color para los globos oculares.

-bd [color] : Selecciona un color para el borde de la ventana.