6
29

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

"CAMPUS ARAGON"

"EDITOR DE PANTALLAS EN AMBIENTE UNIX"

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

C A B R E R A P O M A R C A R L O S

MEXICO, D. F. 1997

A mi Dios,
Quien a causa de él logro el dia
de hoy este sueño, él que nunca
me quito su apoyo y quien siempre
me cargo en momentos dificiles.

A mis padres,
Candido Cabrera y Graciela Pomar
con mi mas grande amor y agradecimiento
por sus esfuerzos y su apoyo. Gracias por
confiar en mi, nunca les fallaré.

A mis hermanos,
Fernando y Alejandro, con cariño y
amor por ser siempre orgullo mio,
gracias por compartir conmigo este
logro.

Con agradecimiento
Al Ing. Manuel Martínez
por su apoyo y su amistad.

Con agradecimiento a la U.N.A.M
Escuela Nacional de Estudios Profesionales "ARAGON"
por permitirme lograr mi desarrollo profesional.

A mi amigo Miguel Ortega y Juan Sosachicatti
por su amistad y entrega incondicional.

A mi novia María Elena Valdez
por su apoyo en los últimos años.

EDITOR DE PANTALLAS EN AMBIENTE UNIX.

ÍNDICE

PAG.

INTRODUCCIÓN. 1

CAPITULO I. ANTECEDENTES. 3

I.1. UNA SESIÓN CON UNIX.. 4
I.2. INICIO DE SESIÓN. 5
I.2.1. Tecleo de comandos. 6
I.2.2. Comportamiento extraño de la terminal. 8
I.2.3. Errores de tecleo. 9
I.3. DETENCIÓN DE UN PROGRAMA. 13
I.4. TERMINACIÓN DE UNA SESIÓN. 13
I.5. CORREO. 14
I.5.1. Envio de mensajes a otros usuarios. 15
I.6. EL MANUAL. 18

CAPITULO II. SISTEMA DE ARCHIVOS. 20

II.1. ARCHIVOS Y COMANDOS COMUNES. 20
II.2. CREACIÓN DE ARCHIVOS: EL EDITOR. 21
II.3. IMPRESIÓN DE ARCHIVOS. CAT Y PR. 27

II.4. MOVIMIENTO, COPIADO Y REMOCIÓN DE ARCHIVOS, MV, CP, RM. **30**
II.5. ¿ CÓMO ES EL NOMBRE DE UN ARCHIVO ? **32**
II.6. ALGUNOS COMANDOS ÚTILES **33**
II.7. DIRECTORIOS **40**

CAPITULO III. INTRODUCCIÓN AL SHELL. **48**

III.1. SUSTITUCIÓN DE METACARACTERES. **50**
III.2. SUBSTITUCIÓN DE CARACTERES. **52**
III.2.1. Variables. **53**
III.3. COMUNICACIÓN ENTRE PROCESOS. **53**
III.3.1. Procesos. **53**
III.4. ABREVIATURAS DE NOMBRES DE ARCHIVOS. **55**
III.5. INTERCONEXIONES (PIPES). **64**
III.6. PROCESOS. **67**
III.7. ADAPTACIÓN AL AMBIENTE. **73**

CAPÍTULO IV. UTILERIAS Y APLICACIONES. **79**

IV.1. INTRODUCCIÓN AL USO DE VENTANAS. **79**
IV.2. CÓMO INICIAR EL PROGRAMA X. **81**
IV.2.1. Ejecución de x (con xdm). **82**

IV.2.2. Inicio del x en una sesión estándar de UNIX. **82**

IV.2.3. El administrador de ventanas. **83**

IV.3. EJECUCIÓN DE PROGRAMAS. **84**

IV.3.1. Para fijar el foco de entrada. **84**

IV.3.2. Cómo trabajar con el ratón. **85**

IV.3.3. La ventana XTERM. **92**

IV.3.4. El menú raiz o principal. **94**

IV.4. OTROS CLIENTES X. **95**

IV.5. FACILIDADES DE ALIAS. **98**

IV.6. FACILIDADES DE HISTORY. **100**

IV.7. UNIX: COMUNICACIONES REMOTAS. **102**

CAPÍTULO V. EDITOR DE PANTALLA (Código Fuente) **109**

CONCLUSIONES **150**

APÉNDICE **152**

BIBLIOGRAFÍA **155**

EDITOR DE PANTALLAS EN AMBIENTE UNIX

Objetivo:

DESARROLLAR UN PROGRAMA EDITOR DE PANTALLA PARA APOYO ACADÉMICO EN UN AMBIENTE UNIX, PARA INCORPORAR LA FUNCIONALIDAD DE UN EDITOR COMERCIAL, ADEMÁS DE PROVEER UNA INTERFAZ INTUITIVA Y AMIGABLE AL USUARIO.

INTRODUCCIÓN

¿ QUE ES UNIX ?

En sentido estricto, es el núcleo (Kernel) de un sistema operativo de tiempo compartido: un programa que controla los recursos de una computadora y los asigna entre los usuarios. Permite a los usuarios correr sus programas; controla los dispositivos periféricos (discos, terminales, impresoras y otros) conectados a la máquina a largo plazo de información tal como programas, datos y documentos.

En sentido más amplio, " UNIX " abarca no sólo el núcleo, sino también programas esenciales; entre ellos; compiladores, editores, lenguajes de comandos, programas para copiado e impresión de archivos, etc.

En un sentido más amplio todavía, "UNIX" puede incluir programas desarrollados por cualquier usuario para ser ejecutados en un sistema; por ejemplo, herramientas para preparar documentos, rutinas para análisis estadísticos y paquetes gráficos.

- El sistema UNIX parece algunas veces más difícil de lo que es (es difícil para un principiante saber aprovechar al máximo las facilidades disponibles. Pero, por fortuna, es fácil empezar a utilizarlas).

El conocimiento de sólo unos cuantos programas debe ser suficiente para dar el primer paso como podrian ser:

- principios básicos: iniciar y terminar una sesión, comandos simples, corrección de errores de tecleo, correo, comunicación entre terminales.

- uso diario: archivos y el sistema de archivos, impresión de archivos, directorios, comandos de mayor uso.

el intérprete de comandos o shell: abreviaturas de nombres de archivos, redireccionamiento de entrada y salida, " pipes " (interconexiones), definición de caracteres de borrado y descontinuado de lineas, y la definición de una trayectoria propia para búsqueda de comandos.

El objetivo de este trabajo este proyecto es desarrollar un programa editor de pantalla en un ambiente multiusuario UNIX, el editor incorporará la funcionalidad de un editor comercial además de proveer una interfaz intuitiva y amigable al usuario.

CAPITULO I. ANTECEDENTES.

Objetivo:

PRESENTAR UNA INTRODUCCIÓN A LO QUE ES EL ENTORNO OPERATIVO UNIX.

CAPITULO I. ANTECEDENTES.

El sistema UNIX es full dúplex: los caracteres que se escriben en el teclado son enviados al sistema, el cual los devuelve a la terminal para que aparezcan en la pantalla. Normalmente, este proceso de eco los copia directamente a la pantalla; de esta manera se puede ver lo que uno esta tecleando aunque en ciertas ocasiones, como cuando se teclea una contraseña secreta ("password"), el eco se apaga de modo de que los caracteres no aparecen en la pantalla.

La mayoria de los caracteres del teclado son caracteres ordinarios sin ningún significado especial, pero algunos le dicen a la computadora como interpretar lo que escribe el usuario. Sin duda, la más importante de las teclas es la tecla RETURN debe ser oprimida antes de que el sistema interprete los caracteres que se acaban de teclear.

RETURN es un ejemplo de un carácter de control, un carácter invisible que controla algunos aspectos referentes a entrada y salida de una terminal. En cualquier terminal común, el carácter RETURN, posee una tecla propia, no asi la mayoria de los caracteres de control. Estos deben ser escritos manteniendo oprimida la tecla CONTROL, llamada también CTL o CNTL o CTRL, y oprimiendo otra tecla, por lo general una letra. Por ejemplo RETURN puede teclearse oprimiendo la tecla RETURN o, en forma equivalente, manteniendo oprimida la tecla CONTROL y tecleando simultáneamente una " m " RETURN puede, por tanto, designarse como **control-m**, lo cual escribiremos

como minúscula **ctl-m**. Entre otros caracteres de control están **ctl-d**, el cual indica un programa que ya no hay más datos de entrada; **ctl-g** que acciona la campana de la terminal; **ctl-h**, frecuentemente llamado " backspace " o retroceso que puede ser usado para corregir errores en la escritura; y **ctl-i**, también llamado tab o tabulador, que avanza el cursor a la siguiente posición del tabulador en forma parecida a lo que sucede en una máquina de escribir. Las paradas del tabulador en los sistemas UNIX están separadas por ocho espacios en la mayoría de las terminales, los caracteres backspace y tab poseen sus propias teclas.

Otras dos teclas también tienen significado especial: **DELETE**, algunas veces llamada RUBOUT o designada con alguna abreviatura, y **BREAK**, también llamada INTERRUPT. En la mayoría de los sistemas UNIX la tecla DELETE termina inmediatamente la ejecución de un programa sin esperar a que acabe en algunos sistemas, **ctl-c** cumple esta misma función. En algunos sistemas, según la forma en que estén conectadas las terminales, BREAK es sinónimo de DELETE o ctl-c.

I.1. UNA SESIÓN CON UNIX.

Al establecer una conexión con una terminal UNIX, por lo general aparecen en pantalla los siguientes mensajes, que no piden otra cosa más que los datos del usuario que se está conectando, esto por seguridad de los propios usuarios del sistema y de sus archivos.

login: you	Teclee su nombre y luego oprima la tecla RETURN.
Password:	Su contraseña no será repetida al irla tecleando.
You have mail.	Hay correspondencia qué leer después de iniciada la sesión.
S	El sistema está listo para recibir ordenes

Sun Sep 25 23:02:57 EDT 1983

I.2. INICIO DE SESIÓN.

Es necesario disponer de alguna clave de usuario y una clave confidencial (password) o contraseña que se puede obtener del administrador del sistema.

Asegúrese que todos los controles estén asignados correctamente en la terminal: letras mayúsculas y minúsculas, full dúplex y otros detalles que sean sugeridos por el encargado del sistema, tales como la velocidad de transmisión (*velocidad en bauds*) . Establezca conexión entre el sistema y la terminal de la forma en que sea necesario, el sistema debe preguntar la clave de usuario mandando el siguiente mensaje: **login:**

Si el sistema envía " basura ", la terminal puede estar a una velocidad de transmisión equivocada; revise el control de velocidad y los otros controles o conmutadores. Si no se resuelve el problema, presione lentamente la tecla BREAK o INTERRUPTOR varias veces. Si no se recibe el mensaje anterior, será necesario pedir asesoría.

Una vez que se ha recibido el mensaje login:, teclee la clave de acceso con *letras minúsculas*. A continuación presione RETURN si es necesario una clave confidencial, le será solicitada y no aparecerá mientras sea tecleada.

La culminación de los esfuerzos por entrar en sesión es un "carácter o símbolo de espera" (*prompt*) consistente usualmente en un solo carácter que indica que el sistema está listo para aceptar comandos del usuario. Por lo general este carácter es un signo de pesos **$** o de porcentaje **%** , pero el usuario puede cambiarlo por alguno de su elección; más adelante mostraremos cómo hacerlo. El carácter es de hecho generado por un programa llamado *interprete de comandos o shell*, el cual es la principal interfaz entre el sistema y el usuario.

Es posible que aparezca un mensaje del día justo antes del carácter de espera, o una notificación de que el usuario tiene correo. También se puede permitir al usuario información acerca del tipo de terminal que está usando; su respuesta permite al sistema utilizar algunas características especiales que ella pudiera tener.

I.2.1. Tecleo de comandos.

Una vez que se haya recibido el símbolo de espera, se puede teclear comandos, los cuales son solicitudes para que el sistema lleve acabo una acción. En adelante usaremos la palabra programa como sinónimo de comando. Cuándo se vea el carácter de espera (supongamos que es **$**) teclee **date** y presione

RETURN. El sistema deberá responder con la fecha y la hora, luego mandará otro carácter de espera. Así pues, la acción completa se vera como sigue en la terminal :

$ date

Mon Sep 26 12:20: 57 DE 1983

$

No olvide el RETURN y no teclee **$.** Si el usuario piensa que esta siendo ignorado, deberá presionar RETURN; algo deberá suceder. RETURN no volverá a ser mencionado otra vez, pero es necesario presionarlo al final de cada linea.

El siguiente comando a ensayar es **Who** , el cual dice quién está en ese momento en sesión:

$ who

rlm	**tty0**	**Sep 26 11:17**
pjw	**tty4**	**Sep 26 11:30**
gerard	**tty7**	**Sep 26 10:27**
mark	**tty9**	**Sep 26 07:59**
you	**ttya**	**Sep 26 12:20**

$

La primera columna corresponde al nombre del usuario. La segunda a su

nombre que el sistema asigna a la conexión que está siendo utilizada (" tty " significa " teletipo ", un antiguo sinónimo de " terminal "). El resto indica cuándo entró el usuario en sesión. Podría intentarse también lo siguiente:

$ who am i
you ttya Sep 26 12:20
$

Si se comete un error al teclear el nombre de un comando y se hace referencia a un comando inexistente, le será advertido que no se encontró ninguno con ese nombre:

$ whom *Nombre de comando mal escrito*
whom: not found *... de modo que el sistema no sabe ejecutarlo*
$

Desde luego si se teclea en forma inadvertida el nombre correcto de un comando, éste se ejecutará aunque con resultados misteriosos.

I.2.2. Comportamiento extraño de la terminal.

Algunas veces la terminal trabajará en forma extraña; por ejemplo, cada letra puede aparecer dos veces o el RETURN quizá no ponga el cursor en la primera columna de la línea siguiente. Por lo general, se puede corregir esto apagando la terminal y volviendo a encenderla, o bien saliendo de la sesión y entrando de nuevo. También se puede leer la descripción del comando **stty**

("definir opciones de terminal") en la sección 1 del manual. Para hacer un tratamiento correcto de los caracteres tabulador ("tab") si la terminal no posee tabs, se teclea el comando

$ stty - tabs

y el sistema convertirá los tabs al número correcto de espacios. Si la terminal tiene paradas del tabulador ajustables por la computadora, el comando **tabs** los ajustará correctamente. (Podría ser necesario teclear

$ tabs ***tipo de terminal***

I.2.3. Errores de tecleo.

Si se comete un error de tecleo y si se da uno cuenta antes de haber oprimido la tecla RETURN, existen dos formas de corregir dicho error: borrar un carácter a la vez o bien descontinuar la línea con el carácter *kill* volviéndola a teclear después.

Si se teclea el carácter de **ctl-u** (descontinuado de linea), por (default) un signo, la linea entera será suprimida como si nunca hubiera sido tecleada, colocando el cursor al principio de una nueva linea:

$ ddtae	*Completamente equivocado; comience*
date	*de nuevo en otra linea*

Mon Sep 26 12:23:39 EDT 1983

S

El carácter # borra el último carácter tecleado; cada vez que se oprime # se borra un carácter de adelante hacia atrás en la línea (sin salirse de ella). En esa forma, si se teclea algo de manera incorrecta, se puede corregir sobre la marcha:

S dd#atte##e	Arréglelo conforme avance
Mon Sep 26 12:24:02 EDT 1983	
S	

Los caracteres de borrado y descontinuado (supresión) de líneas son *sumamente* dependientes del sistema. En muchos sistemas (incluyendo el utilizado por nosotros), el carácter de borrado se ha cambiado por el de retroceso (backspace) ctl-h, el cual da excelentes resultados en terminales de video. Fácilmente se puede comprobar si éste es el caso en nuestro sistema:

S datee-	Intente
datee-: not found	No es
S datee#	Intente #
Mon Sep 26 12:26:08 EDT 1983	Es #

S

(Se ha impreso el carácter retroceso como - para que el usuario pueda identificarlo.) Otra opción usual para el carácter del descontinuado de líneas es **ctl-u**.

¿ Qué sucede si es necesario incluir un carácter de borrado o de descontinuado de línea como parte del texto ? Si los caracteres especiales como # están precedidos por una diagonal invertida \, pierden su significado especial. Asi, para escribir un # o un, se teclea \ # o \ . El sistema podria poner el cursor de la terminal en la siguiente línea después de haber tecleado , aun si éste fue precedido por una diagonal invertida. No hay que preocuparse: el símbolo fue grabado.

La diagonal invertida, también llamada *carácter de escape*, se usa ampliamente para indicar que el siguiente carácter es especial de alguna manera. Para borrar una diagonal invertida, es necesario teclear dos veces el carácter de borrado: \ ## ¿ Entiende por qué ?

Los caracteres que tecleamos son examinados e interpretados por una secuencia de programas antes de llegar a su destino, y la manera exacta en que son interpretados depende no sólo de dónde terminan sino de cómo llegaron ahí.

Cada carácter que se teclea es inmediatamente retransmitido a la terminal, a menos que la retransmisión (eco) esté suspendida, lo cual no es muy usual. Los caracteres son almacenados temporalmente por el núcleo hasta que se oprima RETURN, de esta forma los errores de tecleo pueden corregirse con los caracteres de borrado y descontinuado de líneas. Cuando uno de estos caracteres está precedido por una diagonal invertida, el núcleo la descarta y almacena el carácter siguiente sin interpretación.

Cuando se oprime la tecla RETURN, los caracteres almacenados se envían al programa que está leyendo datos de la terminal. Dicho programa puede, en un momento dado, interpretar en una forma especial los caracteres recibidos; por ejemplo, el shell no hace ninguna interpretación especial de un carácter si éste está precedido por una diagonal invertida. Volveremos a hablar de esto en el capítulo 3. Por lo pronto, debe recordarse que el núcleo procesa los caracteres de borrado y descontinuado de líneas, y la diagonal invertida sólo si ésta se encuentra antes de dichos caracteres; después de esto cualquier carácter que quede puede ser interpretado también por otros programas.

El núcleo lee lo que se teclea mientras el usuario lo está tecleando, aun si se encuentra ocupado con alguna otra cosa, así, puede teclearse tan rápido como se quiera y cuando se quiera, aun cuando algún comando se esté desplegando. Si se teclea mientras el sistema está desplegado, los caracteres de entrada aparecerán mezclados con los de salida, pero serán almacenados e interpretados en el orden correcto. Es posible teclear comandos uno después de otro sin

esperar a que terminen, o aun sin que hayan comenzado.

I.3. DETENCIÓN DE UN PROGRAMA.

Se puede interrumpir la ejecución de la mayor parte de los comandos tecleando el carácter DELETE. La tecla BREAK, que se encuentra en la mayoría de las terminales, también puede servir, aunque esto depende del sistema. En ciertos programas tales como los editores de textos, DELETE interrumpe lo que se esté haciendo el programa sin sacarnos de él. Al apagar la terminal o colgar el teléfono se detendrá la mayoría de los programas.

Si sólo se desea parar momentáneamente la salida de un programa, (por ejemplo, para evitar que algo muy importante desaparezca de la pantalla), teclee *ctl-s*. La salida se detendrá casi inmediatamente; el programa será suspendido hasta que lo reiniciemos. Cuando se quiera continuar, se teclea *ctl-g*.

I.4. TERMINACIÓN DE UNA SESIÓN.

La manera adecuada de terminar una sesión es teclear *ctl*-d en vez de un comando: esto indica que ya no hay más entradas. Frecuentemente sólo es necesario apagar la terminal o colgar el teléfono, pero depende de nuestro

sistema que esto realmente termine una sesión.

I.5. CORREO.

El sistema proporciona un sistema de correo para comunicarse con otros usuarios, de modo que alguna vez al iniciar una sesión se verá el mensaje

You have mail.

antes del primer simbolo de espera. Para leer el correo si teclea

$ mail

El correo será impreso, un mensaje a la vez, empezando por el más reciente. Después de cada párrafo, mail esperará que se decida qué hacer a continuación.

Las dos respuestas básicas son d, para borrar el mensaje, y RETURN para dejarlo (por lo que volverá a aparecer dicho mensaje la siguiente vez que se vea el correo).

Otras posibles respuestas son **p** para reimprimir un mensaje, **s** nombre de archivo para almacenar el mensaje en el archivo nombrado y **q** para salir del

comando **mail** .

Enviar correo a alguien es una cosa sencilla. Supóngase que el correo es para una persona cuya clave de acceso es nico. La manera más fácil de enviarlo es ésta:

S mail nico

Ahora teclee el texto de la cara empleando tantos renglones como desee . . .

Después del último renglón de la carta teclee un control-d

ctl-d

S

El *ctl*-d señala el final de la comunicación indicando al comando **mail** que ya no hay más entradas. Si uno cambia de opción a la mitad de la comunicación, se presiona DELETE en lugar de *ctl*-d. La comunicación incompleta será almacenada en un archivo llamado **dead**. leter en vez de ser enviada.

Existen otras maneras de enviar correo; podemos enviar una carta preparada previamente o enviarla a varias personas a la vez, e incluso mandarla a gente que trabaja en otras máquinas.

1.5.1. Envío de mensajes a otros usuarios.

Si el sistema UNIX posee múltiples usuarios, alguna vez podria aparecer en la

terminal algo como esto

Message from mary tty?

acompañado de una señal audible. Mary desea un mensaje, pero no se podrá conectar a menos que teclee

S write mary

Esto establece un sistema de comunicación bidireccional. Ahora las líneas que teclea Mary en su terminal aparecerán en la nuestra y viceversa, aunque esto se hará de manera lenta como si el usuario se comunicara con algo muy lejano.

Si el usuario está realizando una tarea es necesario llegar a un estado en el cual pueda teclear un comando. Normalmente, cualquier programa que se esté ejecutando tiene que detenerse por sí mismo o ser detenido, pero algunos programas, como el editor y el mismo **write**, tiene un comando '!' para retorno temporal al shell

El comando **write** no impone reglas, por lo que es necesario un protocolo para impedir que lo que uno teclea se confunda con lo teclea Mary. Una convención es turnarse, terminando cada turno con (c), que significa ' cambio ', o bien indicar el deseo de terminar la conversación con (cf), que significa ' cambio y fuera '.

Terminal de mary:	*Terminal del usuario:*
S write mary	**S Message from mary tty?...**
Message from you ttya...	**did you forget lunch?** (o)
did you forget lunch? (o)	**five**
	ten minutes (o)
ten minutes (o)	
ok (o o)	**ok (o o)**
	ctl-d
EOF	
ctl-d	
	S EOF

Otra forma de salirse del comando write es presionar DELETE. Obsérvese que los errores de tecleo no aparecen en la terminal de Mary.

Si se intenta enviar un mensaje a alguien que no se encuentra en sesión o que no quiere ser interrumpido, la máquina lo hará saber. Si la otra terminal se encuentra en sesión pero no se recibe respuesta después de un tiempo razonable, puede significar que la persona a la que queremos mandar el mensaje puede estar ocupada o alejada de la terminal; habrá entonces que teclear *ctl*-d o.

DELETE. Si no se desea ser interrumpido, entonces habrá que usar **mesg (l).**
News

Muchos sistemas UNIX proporcionan un servicio de "noticias", el cual mantiene al tanto a los usuarios acerca de eventos de mayor o menor interés. Teclee

$ news

Existe asimismo una gran red de sistemas UNIX, los cuales se mantienen en contacto a través de llamadas telefónicas.

I.6. EL MANUAL.

El manual del programador de UNIX describe la mayor parte del material necesario para familiarizarse con el sistema. La sección 1 trata de los comandos. El resto de las secciones tratan de funciones para ser usadas por programadores en C, formatos de archivos y mantenimiento del sistema. (La numeración de estas secciones varía de sistema a sistema). No olvide consultar el índice invertido al principio; puede hojearse rápidamente en busca de comandos que pueden ser útiles para lo que se intenta hacer. También se encuentra una introducción al sistema que proporciona un resumen sobre su funcionamiento.

Frecuentemente, el manual es mantenido en línea de manera que pueda consultarse en una terminal. Si alguna cosa se llegará a complicar y no hay un asesor cerca, es posible imprimir cualquier página del manual en la terminal con el comando who, debe teclearse

$ man who

y por su puesto

$ man man

da una explicación del comando man

Instrucción asistida por computadora

Es muy probable que el sistema en el que el usuario se encuentre trabajando posea un comando llamado **learn** , el cual proporciona una instrucción asistida por computadora sobre el sistema de archivos y comandos básicos, el editor, preparación de documentos y aun sobre programación del lenguaje C. Inténtese lo siguiente

$ learn

Si el comando learn existe en el sistema, éste dirá qué hacer a continuación. Si dicho comando falla, deberá ensayarse el comando teach.

Para ver una referencia rápida de algunos de comandos más utilizados dentro del entorno UNIX vea el apéndice.

CAPITULO II. SISTEMA DE ARCHIVOS.

Objetivo:

IDENTIFICAR LA MANERA COMO ESTÁN ESTRUCTURADOS LOS DIRECTORIOS Y LOS ARCHIVOS DENTRO DEL SISTEMA OPERATIVO UNIX.

CAPITULO II. SISTEMA DE ARCHIVOS.

II.1. ARCHIVOS Y COMANDOS COMUNES.

En un sistema UNIX, la información es almacenada en *archivos* , los cuales son muy semejantes a los archivos ordinarios de una oficina. Cada archivo tiene un nombre, un contenido, un lugar para almacenarlo y cierta información de tipo administrativo; por ejemplo: quién es el dueño, y qué tan grande es. Un archivo puede conectarse a una carta, una lista de nombres y direcciones, las instrucciones fuente de un programa, datos para ser usados por un programa e incluso programas en su forma ejecutable, así como otro tipo de material que no esté en forma de texto.

El sistema de archivos de UNIX está organizado en tal forma que sea posible mantener archivos personales sin interferir en los que pertenezcan a otros y evitar que otra gente interfiera con dichos archivos. Existe una gran cantidad de programas que manipulan archivos, pero por ahora nos ocuparemos sólo de aquellos que son los más frecuentemente usados.

II.2. CREACIÓN DE ARCHIVOS: EL EDITOR.

Si se desea teclear un artículo, una carta o un programa, ¿ Cómo puede almacenarse la información en la máquina ? La mayor parte de estas tareas se llevan acabo con un editor de textos, el cual es un programa para almacenar y manipular información dentro de la computadora. La mayor parte de los sistemas UNIX poseen un editor de pantalla, un editor que aprovecha las ventajas de las terminales modernas para mostrar los efectos que los cambios producen en el contexto, mientras se efectúan. Dos de los más populares son **vi** y **emacs**. Sin embargo, no describiremos aqui especificamente ninguno de estos editores de pantalla, por una parte a causa de las limitaciones tipográficas y por otra a que no existe un editor estándar.

Existe, sin embargo, un editor más antiguo llamado **ed** que seguramente estará disponible en todos los sistemas. No aprovecha ninguna caracteristica especial de la terminal, por lo que funcionará en cualquiera de ellas. También es fundamento de otros programas esenciales (incluyendo algunos editores de pantalla), por lo que será importante que se aprenda tarde o temprano.

Para crear un archivo llamado **junk** con algún texto dentro de él, usando **ed** deberá procederse como sigue:

$ ed	*Invoca al editor de textos*
a	*Comando de ed para agregar texto*

Ahora teclee

el texto que quiera

. *Teclee un '. ' para dejar de añadir texto.*

w junk *Escriba tu texto a un archivo llamado junk*

39 ***de** imprime el número de caracteres escritos*

q *Abandone de*

S

El comando a ("anexar") ordena a **ed** que empiece a recibir el texto. El "." que delimita el final del texto debe ser tecleado al principio de la línea (sin ningún otro carácter). Es importante recordar esto pues, mientras no se teclee ":", ningún otro comando será reconocido (cualquier cosa que se teclee será tratado como algo que debe añadirse).

El comando **w** del editor ("escribir") almacena la información que a sido tecleado por el usuario; "**w junk**" la almacena en un archivo llamado **junk**. El nombre del archivo puede ser cualquier palabra que se desee; hemos escogido **junk** ("basura") para señalar que este archivo no es muy importante.

ed responde con el número de caracteres que escribió en el archivo. Antes de que el comando w sea invocado, ninguna información se almacena en forma

permanente, por lo que si uno termina la sesión, la información no se guarda en el archivo. (Si se abandona la sesión durante la edición, los datos con los cuales se estaba trabajando se almacenan en un archivo llamado **ed.hup**, con el cual se puede continuar trabajando en la siguiente sesión). Si el sistema llegará a "caerse" (es decir, que llegue a detenerse de manera inesperada debido a alguna falla de *hardware o de software)* durante la edición, el archivo sólo contendrá la información que fue salvada por el último comando de "escritura". Después de teclear **w** la información se almacena en forma permanente; se puede accesar posteriormente tecleando

S de junk

Por su puesto, el texto que ha sido tecleado puede editarse para corregir errores mecanográficos, alterar la redacción, rearreglar párrafos y otras cosas por el estilo. Cuando se ha concluido, el comando **q** ("salir") termina la sesión de edición.

¿ Con qué archivos se cuenta ?

El contador de caracteres de **ed** incluye el carácter que se encuentra al final de cada linea, llamado *newline* el cual es la forma en que el sistema representa RETURN el comando **ls** lista los nombres (pero no el contenido de los archivos:

S ls

junk

temp

S

Que en efecto es el archivo recién creado. (Podría también haber otros que no fueron creados por el usuario como por ejemplo temp). Los nombres de los archivos se ordenan automáticamente por orden alfabético.

ls, al igual que la mayoría de los comandos, tiene opciones que pueden ser usadas para alterar su comportamiento normal. Dichas opciones siguen al nombre del comando en la línea de comandos, y por lo general están compuestos por un signo menos '_' inicial, seguido de una letra que hace referencia a su significado. Por ejemplo, **ls_t** nos permite obtener un listado de los archivos por orden de "antigüedad" **(time),** es decir, según la fecha en la que hayan sido alterados por última vez, empezando con el más reciente.

S ls-t

temp

junk

La opción -l ("long") nos permite obtener un listado más completo que ofrece mayor información sobre cada archivo:

S ls -l

total 2

-rw-r--r-- 1 you 19 Sep 26 16:25 junk

-rw-r--r-- 1 you 22 Sep 26 16:26 temp

"Total 2" dice cuántos bloques de espacio en disco están ocupando los archivos;

un bloque se compone generalmente de 512 o 1024 caracteres. La cadena **-rw-r--r--** indica quien tiene autorización para leer o escribir en el archivo; en este caso el dueño **(you)** puede tanto leer como escribir pero otros usuarios únicamente pueden leerlo. El "1" que se encuentra a continuación es el número de ligas con el archivo; "you" es el dueño del archivo, esto es, la persona que lo creó. 19 y 22 son el número de caracteres en cada archivo los cuales coinciden con los generados por **de** . La fecha y la hora indican cuando fue alterado por última vez el archivo.

Las opciones pueden ser agrupadas: **ls-lt** proporcionan los mismos datos que **ls-l** , pero ordenados con los archivos más recientes al principio. La opción -u da información acerca de cuando fueron usados los archivos: **ls-lut** genera un archivo completo (-l) de archivos, ordenados de acuerdo con el uso más reciente. La opción **-r** invierte el orden de la salida; así **ls-rt** los lista por el orden de uso menos reciente también es posible hacer referencia sólo a ciertos archivos de interés, y **ls** listará únicamente información acerca de ellos:

S ls -l junk

-rw-r--r-- 1 you 19 Sep 26 16:25 junk

S

Las cadenas que siguen al nombre del programa en la línea de comandos, tales como -l y junk en el ejemplo anterior, se llaman argumentos del programa. Los argumentos son por lo general opciones o nombres de archivos a ser usados por el comando .

El hecho de especificar opciones con un signo menos y una letra, tal como **-to** la combinación **-lt** es una convención común.

En general, si un comando acepta dichos argumentos opcionales, éstos proceden a cualquier argumento de nombre de archivo, pero es posible también que aparezcan en cualquier otro orden. Sin embargo, los programas en UNIX son caprichosos en su tratamiento de las opciones múltiples. Por ejemplo, la séptima edición estándar de **ls** no aceptará

S ls -l -t *No funciona en la séptima edición*

Como sinónimo de ls y -lt, mientras que otros programas requerirán que las opciones múltiples estén separadas.

Cada comando tiene sus propias peculiaridades y su propia elección del significado de cada letra (que a veces difiera de la misma función en otros comandos). Este comportamiento impredecible es desconcertante y se considera algunas veces como el mayor defecto del sistema. Aunque esa situación está siendo corregida (las nuevas versiones poseen mayor uniformidad) lo único que podemos aconsejar es escribir con mucho esmero los programas propios.

II.3. IMPRESIÓN DE ARCHIVOS, CAT Y PR.

Ahora que ya se tienen algunos archivos, ¿ Cómo podemos observar su contenido ? exísten muchos programas que nos permiten hacerlo, tal vez más de los que sean necesarios. Una posibilidad es usar el editor:

$ de junk

19	*de comunica que hay 19 caracteres en* **junk**
1,$p	*Imprime de la linea 1 a la última*
To be or not to be	*El archivo tiene una sola línea*
q	*Todo terminado*
$	

ed comienza comunicando el número de caracteres en junk; el comando **1,$p** indica a ed que imprima o despliegue todas las líneas existentes en el archivo. Una vez que el lector aprenda cómo usar el editor, podrá ser selectivo en cuanto a lo que se manda a imprimir.

Hay ciertas ocasiones en las que no es factible usar un editor para imprimir o desplegar. Por ejemplo, existe un límite del tamaño de un archivo que puede ser manejado por **ed** (algunos miles de líneas). Además este sólo imprimirá un archivo a la vez, y algunas veces se necesitará imprimir varios, uno después de

otro sin detenerse. A continuación mencionaremos dos posibles alternativas.

La primera es **cat** , el más simple de todos los comandos de impresión. cat imprime el contenido de todos los archivos referenciados por sus argumentos:

$ cat junk

To be or not to be

$ cat temp

That is the question.

$ cat junk temp

To be or not to be

That is the question.

$

El archivo o archivos referenciados son concatenados (de aquí el nombre "cat") en la terminal, uno después del otro sin dejar espacio entre ellos.

No hay ningún problema con los archivos cortos, pero con los extensos se tendrá que usar rápidamente **ctl-s** si se tiene una conexión de alta velocidad con la computadora para detener la salida de **cat** antes de que se pierda de la pantalla. No existe un comando "estándar" para desplegar un archivo en una terminal de video una pantalla por vez, aún cuando la mayoria de los sistemas UNIX lo tienen.

Al igual que cat, el comando pr imprime el contenido de todos los archivos referenciados en una lista, pero en una forma apropiada para impresoras de líneas: cada página tiene 66 líneas (11 pulgadas), con la fecha y la hora en que el archivo fue alterado, el número de página y el nombre del archivo al principio de cada página, así como renglones en blanco adicionales para no imprimir en el dobles de la hoja. Asi pues, para imprimir **junk** nitidamente, después saltar al principio de la siguiente hoja e imprimir **temp** nitidamente:

$ pr junk tem

Sep 26 16:25 1983 junk Page 1

To be or not to be

(60 líneas más en blanco)

Sep 26 16:26 1983 temp Page 1

That is the question.

(60 líneas más en blanco)

$

pr puede generar también una salida en varias columnas:

$ pr -3 archivos

Imprime cada archivo en un formato de 3 columnas. Puede usarse cualquier número razonable en lugar de "3" y pr hará lo más que se pueda. (en lugar de la palabra *nombres de archivos* habrá que poner en la lista de los nombres de los

archivos).**pr-m** imprimirá un conjunto de archivos en columnas paralelas.
Nótese que **pr** no es un programa para dar formato en el sentido de reordenar líneas y alinear márgenes. Los verdaderos "formateadores" son **nroff y troff,** existen también comandos que imprimen archivos en impresoras de alta velocidad. Búsquense en el manual bajo nombres ales como **lp y lpr,** bien búsquense la palabra "impresora" en el índice invertido. El comando que debe usarse dependerá de que equipo este conectado a la máquina. **pr** y **lpr** se usan juntos con frecuencia; después de que **pr** formatea la información adecuadamente, **lpr** la envía a la impresora de líneas. Volveremos sobre este punto más adelante.

II.4. MOVIMIENTO, COPIADO Y REMOCIÓN DE ARCHIVOS, MV, CP, RM.

El primer problema es cambiar el nombre a un archivo. El cambio de nombre se lleva acabo "moviéndolo" de un nombre a otro, como se muestra a continuación:

S mv junk precious

Esto significa que el archivo que se llamaba junk ahora se llama precious; el contenido no ha sido alterado. Si se ejecuta ls ahora, se notará un cambio en el listado: junk ya no se encuentra ahí pero precious sí.

S ls

precious

temp

S cat junk

cat: can't open junk

S

Tómese en cuenta que, si se mueve un archivo a otro que ya existía, el segundo archivo será reemplazado.

Para generar una copia de un archivo (esto es, para tener dos versiones de algo). se usa el comando cp:

S cp precious precious.save

generar una copia de precious en precious. save.

Finalmente, el comando rm remueve todos los archivos a los que se haga referencia:

S rm temp junk

rm: junk nonexistent

S

Si alguno de los archivos que debían eliminarse no existiera, el sistema enviará mensajes de advertencia, pero de lo contrario, rm llevará acabo su tarea sin enviar ningún mensaje, al igual que la mayoría de los comandos de UNIX. No

hay ninguna señal, y los mensajes de error son breves y, algunas veces, no muy útiles. Dicha brevedad puede ser desconcertante para los principiantes, pero los usuarios expertos consideran molestos a los comandos que dan mucha información.

II.5. ¿ CÓMO ES EL NOMBRE DE UN ARCHIVO ?

Hasta aquí hemos usado nombres de archivos sin decir cómo deben ser construidos, por lo cual mencionaremos ahora un par de reglas. Primera, los nombres de archivos están limitados a 14 caracteres. Segunda, aunque en un nombre de archivo puede utilizarse casi cualquier carácter, el sentido común indica que conviene utilizar los que son visibles y que deben evitarse los que puedan utilizarse con otro significado. Hemos visto, por ejemplo, que en el comando ls, la opción ls-t genera un listado de los archivos ordenando según el tiempo que haya pasado después de modificarlos por última vez. Por lo tanto, si el lector tiene un archivo llamado -t, será difícil poder listarlo por nombre. (Cómo podría hacerse esto ?) Además del signo menos como primer carácter, existen otros con significado especial. Para evitar problemas se aconseja usar únicamente letras, números, el punto y la línea de subrayado, (El punto y el subrayado se usan, por convención, para dividir los nombres de archivos en partes, como en precious. save en el ejemplo anterior). Finalmente, es importante no olvidar la diferencia entre mayúsculas y minúsculas (algo, Algo y ALGO son tres nombres diferentes).

II.6. ALGUNOS COMANDOS ÚTILES.

Para concretar la discusión usaremos un archivo llamado poem, el cual contiene un conocido verso de Augustus De Morgan. Creémoslo usando ed:

```
$ ed
a
Great fleas have little fleas
upon their backs to bite ' em,
And little fleas have lesser fleas
and so ad infinitum.
And the great fleas themselves, in turn,
have greater fleas to go on;
While these again have greater still,
and greater still, and so on.
.
w poem
263
q $
```

El primer comando cuenta las líneas, palabras y caracteres en uno o varios archivos; su nombre es wc (cuenta palabras):

```
$ wc poem
       8    46    263  poem
$
```

Lo anterior significa que poem tiene 8 líneas, 46 palabras y 263 caracteres. La definición de "palabra" es muy simple: cualquier cadena de caracteres que no contengan blancos o alguno de los caracteres de tabulación o nueva- línea.

wc también puede contar más de un archivo (e imprimir los totales), así como suprimir algún conteo si así se requiere.

En el segundo comando se llama **grep;** éste busca en un archivo las líneas que se ajusten a un patrón. (El nombre proviene del comando de ed:g/ regular-expresión/p). Supóngase que se desea buscar la palabra "fleas" en el archivo poem:

```
$ grep fleas poem
Great fleas have little fleas
And little fleas have lesser fleas,
And the great fleas themselves, in turn,
have  greater fleas to go on;
$
```

grep también puede buscar líneas que no contengan dicho patrón, cuando se emplea la opción -v. (Su nombre proviene del comando del editor; puede pensarse en este comando como si invirtiera el patrón).

$ grep -v fleas poem

upon their backs to nite ' em,

and so ad infinitum.

While these again have greater still,

and greater still, and on.

$

grep puede usarse para búsqueda en varios archivos; en el caso antepondrá el nombre del archivo a cada línea en que se encuentre la cadena en cuestión, por lo que se podrá saber en qué archivo se encontró dicha línea. Existen también opciones para contar, numerar, etc. El comando **grep** es capaz de manejar cadenas mucho más complicadas que simples palabras tales como "fleas".

El tercer comando en cuestión es **sort**, el cual ordena alfabéticamente la entrada, línea por línea. Esto no es muy interesante para nuestro ejemplo del poema, pero apliquémoslo de todas manera para conocer su funcionamiento:

$ sort poem

and greater still, and so on.

and so ad infinitum.

have greater fleas to go on;

upon their backs to bite ' em,

And little fleas have lesser fleas,

And the great fleas themselves, in turn,

Great fleas have little fleas

While these again have greater still,

S

El ordenamiento se realiza línea por línea, aunque el orden por omisión (default) coloca primero los blancos, luego las letras mayúsculas y después las minúsculas por lo que no es un orden estrictamente alfabético.

sort tiene muchísimas opciones para controlar el orden de la clasificación (orden inverso, orden numérico, orden de diccionario, no tener en cuenta los blancos iniciales, ordena campos dentro de las líneas, etc.), aunque lo más recomendable es ver cómo operan estas opciones para estar seguros de su funcionamiento. He aquí algunas de las más comunes:

sort -r *Invierta orden normal*
sort -n *Clasifique en orden numérico*

sort -nr *Clasifique en orden numérico inverso*
sort -f *Considere igual las mayúsculas y minúsculas*
sort +n *Clasifique comenzando en el campo n + 1 -st*

Otro comando útil para examinar archivos es **tail**, el cual despliega las últimas 10 líneas del archivo. Esto parece ser más excesivo para nuestro poema de ocho líneas; sin embargo, es muy útil para archivos más grandes. Además, **tail** posee una opción para especificar el número de líneas. Así, para desplegar la última línea de poem:

$ tail -1 poem
and greater still, and so on.
$

tail puede usarse también para listar un archivo comenzando con una línea específica:

$ tail +3 archivo

Empieza a desplegar en la tercera línea. (Obsérvese la inversión natural de la convención del digno menos para argumentos).

Los últimos dos comandos sirven para comparar archivos. Supóngase que

tenemos una variante de **poem** en el archivo **new_poem**:

S cat poem

Great fleas have little fleas

upon their backs to bite ' em,

And little fleas have lesser fleas,

and so ad infinitum.

And the great fleas themselves, in turn,

have greater fleas to go on;

While these again have greater still,

and greater still, and so on.

S cat new_poem

Greater fleas have little fleas

upon their backs to bite them,

And little fleas have lesser fleas,

ans so on ad infinitum.

And the greater fleas themselves, in turn,

have greater fleas to go on;

While these again have greater still,

and greater still, and so on.

S

No existe mucha diferencia entre ambos archivos; de hecho, será necesario cierto esfuerzo para encontrarla. Aquí es donde están en acción los comandos para comparar archivos **.cmp** encuentra la primera posición en donde dos archivos difieren:

```
$ cmp poem new_poem
poem new_poem differ: char 58, line2
$
```

Esto quiere decir que los dos archivos difieren en la segunda línea, lo cual es cierto, aunque no aclara cuál es la diferencia ni tampoco identifica las diferencias subsecuentes.

El otro comando para comparar archivos es **diff**, el cual comunica todas las líneas cambiadas, añadidas o borradas:

```
$ diff poem new_poem
2c2
< upon their backs to bite em,
---
> upon their backs to bite them,
4c4
< and so ad infinitum.
---
> and so on ad infinitum.
```

s

Esto quiere decir que la línea 2 en el primer archivo (poem) tiene que cambiarse por la línea 2 del segundo archivo (new_ poem), y la forma similar para la cuarta línea.

En general, se usa **cmp** cuando se desea estar seguro de que dos archivos realmente tiene el mismo contenido. Es rápido y funciona para cualquier tipo de archivo, no sólo para archivos de texto. Se usa **diff** cuando se espera que los archivos sean diferentes y se desea saber exactamente cuáles líneas son las que difieren. diff maneja únicamente archivos de textos.

II.7. DIRECTORIOS.

El sistema puede distinguir el archivo llamado **junk** de los otros que tengan el mismo nombre. La distinción se hace agrupando los archivos en *directorios*, en forma semejante a como se acomodan los libros en estantes en una biblioteca, por lo que archivos en distintos directorios pueden tener el mismo nombre sin ningún conflicto.

Por lo general, cada usuario tiene un directorio personal o "*directorio de origen*", también llamado "directorio de login", el cual contiene únicamente los archivos y directorios de cada usuario en particular.

Archivos que le pertenecen. Cuando uno entra en sesión, se encuentra dentro de

su directorio de origen. Se puede cambiar el directorio en el que se encuentra uno trabajando (a veces llamado directorio de trabajo o directorio actual) pero su directorio de origen es siempre el mismo. A menos que se haga otra cosa, cuando se genera un nuevo archivo éste se crea en el directorio actual. Debido a que éste es inicialmente el directorio de origen, este archivo no tiene ninguna relación con otro del mismo nombre que pudiera existir en el directorio de otra persona.

Un directorio puede contener otros directorios, así como archivos ordinarios ("Los directorios grandes tienen directorios más pequeños..."). La manera natural de visualizar esta organización es considerarla como un árbol de directorios y archivos. Es posible desplazarse dentro de este árbol y encontrar cualquier archivo en el sistema comenzando en la raíz y moviéndose **a través de las ramas adecuada.**

En forma inversa, se puede empezar desde donde uno se encuentra y moverse hacia la raíz.

Intentemos en primer lugar esto último. Nuestra herramienta básica es el comando **pwd** ("muestra de directorio de trabajo"), el cual despliega el nombre del directorio en el que se encuentra uno al momento de teclearlo:

$ pwd

/usr/you

S

Esto significa que el usuario se encuentra actualmente en el directorio **you**, en el directorio **usr**, el cual está a su vez en el *directorio raiz* (root) llamado por convención simplemente '/'. Los caracteres / separan los componentes de un nombre; el límite de 14 caracteres mencionando anteriormente se aplica a cada uno de los componentes de dicho nombre. En muchos sistemas, / **usr** es un directorio que contiene los directorios de todos los usuarios normales del sistema. (Aun si el directorio de origen no es /usr/you, pwd desplegará algo parecido).

A continuación pruébese lo siguiente

S ls /usr

Esto debe desplegar una largo lista de nombres, entre los cuales se encuentra el propio directorio de origen del usuario you.

El siguiente paso es intentar listar la raiz. Deberá obtenerse una respuesta similar a ésta:

S ls /

bin

boot

unix

usr

S

.La mayoría de estos son directorios, aunque unix es realmente un archivo que contiene la forma ejecutable del núcleo de UNIX.

S cat /usr/you/junk

. El nombre

/usr/you/junk

se llama la *trayectoria* (pathname) del archivo. "Trayectoria" tiene un significado intuitivo: representa el nombre completo de una ruta desde la raíz, a través del árbol de directorios, hasta un archivo particular. Es una regla universal en el sistema UNIX que, siempre que pueda usarse un nombre de archivo ordinario, también podrá utilizarse una trayectoria.

El sistema de archivos se encuentra estructurado como un árbol genealógico; el diagrama que se presenta a continuación aclarará esto.

Las trayectorias no serían muy útiles si todos los archivos de interés estuvieran en el directorio del lector, pero si se trabaja con alguna otra persona o en varios proyectos de modo conjunto, entonces se vuelven muy útiles. Por ejemplo, otras personas pueden imprimir el archivo junk del lector tecleando

S cat /usr/you/junk

Asimismo, es posible conocer qué archivos tiene Mary tecleando

S ls /usr/mary

data

junk

S

o bien sacar una copia de alguno de sus archivos por medio de
S cp /usr/mary/data data

o editar su archivo:
S de /usr/mary/data

Si Mary no desea que otra gente maneje sus archivos, o viceversa, puede asegurarse la privacía. Cada archivo o directorio tiene permisos de lectura, escritura o ejecución para el dueño, un grupo o cualquier persona, que pueden usarse para controlar el acceso (Recuérdese ls-1). En nuestros sistemas locales, la mayoría de los usuarios prefieren que sus archivos sean públicos en vez de privados; sin embargo el lector quizá trabaje en un sistema con una política distinta a ésta.
S ls /bin/usr/bin

Cuando se ejecuta un comando tecleando su nombre después del símbolo de espera, el sistema busca un archivo con ese nombre. Normalmente busca primero en el directorio actual de trabajo del usuario (donde probablemente no

lo encuentre), después busca en /bit, y por último en /usr/bin. Nada de especial tiene los comandos como cat o ls, excepto que han sido reunidos en un par de directorios para ser fácilmente localizados y administrados. Para verificar esto, trátese de ejecutar alguno de estos programas más usando sus trayectorias completas:

S /bin/date
Mon Sep 26 23:29:32 EDT 1983
S /bin/who
srm tty1 Sep 26 22:20
cvw tty4 Sep 26 22:40
you tty6 Sep 26 23:04
S

Cambio de directorio, cd

Si se trabaja regularmente con Mary o con información en el directorio de ella es posible decir: "Yo quiero trabajar en los archivos de Mary no en los mios. "Esto puede hacerse cambiando el directorio actual con el comando cd:
S cd /usr/mary

Ahora bien, cuando se use un nombre de archivo (sin diagonales) como argumentos para cat o pr, éste se referirá al archivo en el directorio de Mary. El cambio de directorio no afecta ningún permiso asociado con un archivo (si el

lector no podía accesar un archivo desde su propio directorio, el cambio a otro directorio no alterará esta situación).

Es conveniente por lo general acomodar los archivos de manera que todos los relacionados con alguna cosa queden en un directorio separado de otros proyectos.

Por ejemplo, si se desea escribir un libro, sería deseable almacenar todo el texto en un directorio llamada book. El comando mkdir crea un nuevo directorio.

S mkdir book	*Cree un directorio*
S cd book	*Diríjase a él*
S pwd	*Cerciórese de que está en el sitio adecuado*
/urs/you/book	
...	*Escriba el libro (transcurren varios minutos)*
S cd ..	*Suba un nivel el sistema de archivo*
S pwd	
/usr/you	
S	

'..' se refiere al padre del directorio en el que se encuentre uno trabajando, que ocupa un nivel más cercano a la raíz, '.' es un sinónimo del directorio actual.

S cd *Retorne al directorio de origen*

retornará al usuario a su directorio de origen, el directorio con el cual se inició la sesión.

Para eliminar el directorio book, remuévanse primero todos sus archivos (más adelante mostraremos una forma más rápida) y después úsese cd para trasladarse al directorio padre de book y tecléese

$ rmdir book

rmdir únicamente eliminará un directorio si está vacio:

CAPITULO III. INTRODUCCIÓN AL SHELL.

Objetivo:

CONOCER COMO FUNCIONA EL SHELL DE UNIX, ASÍ COMO SUS PRINCIPALES CARACTERÍSTICAS Y COMANDOS UTILIZADOS DENTRO DE EL.

CAPITULO III. INTRODUCCIÓN AL SHELL

Cuando el sistema despliega el símbolo de espera S y se teclean comandos que son ejecutados por el sistema, no es el núcleo el que se está dirigiendo al usuario, sino un intermediario llamado intérprete de comandos o *shell* . El shell es solamente un programa como **date** o **who**, aunque es capaz de hacer algunas cosas muy interesantes. El hecho de que shell se encuentre entre el usuario y las facilidades del núcleo tiene muchas ventajas; algunas de las cuales explicaremos a aquí. Existen tres de mayor importancia:

· Las abreviaturas de nombres de archivos: se pueden tomar todo un conjunto de nombres de archivos como argumentos de un programa especificado el patrón de los nombres; el shell encontrará los nombres de archivos que igualen dicho patrón.

· Redireccionamiento de entrada-salida: se puede lograr que la salida de cualquier programa se dirija hacia un archivo en lugar de dirigirla hacia una terminal; se puede hacer que la entrada provenga también de un archivo y no de la terminal.

· Personalizar el entorno: el usuario puede definir sus propios comandos y abreviaturas.

El intérprete de comandos llamado shell es el canal de comunicación más importante entre el sistema y sus usuarios. El shell no es parte del sistema

operativo y no goza de privilegios especiales. Es la llave para coordinar y cambiar los programas de Unix. El shell funciona como intermediario entre el usuario y el Kernel (núcleo del sistema operativo). El shell puede actuar de varias formas, como son:

i) Un programa para aceptar y procesar comandos y programas de usuarios.

ii) Substitución textual de variables y cadenas

iii) Un medio para transferir y redireccionar la salida entre archivos y procesos

iv) Procesos asíncronos

v) Visto como lenguaje de programación contiene estructuras de control, paso de parámetros, sustitución de variables y de cadenas de caracteres, procesamiento de archivos de comandos, etc.

vi) Control del ambiente

El shell es, al mismo tiempo, un lenguaje de alto nivel y un lenguaje estructurado: esto es, capaz de proporcionar otra alternativa para los lenguajes de programación convencionales de alto nivel. Proporciona un ambiente de programación y es un medio para realizar llamadas del sistema al Kernel. Además, puede obtener información de tareas, procesos y usuarios.

No reside en forma permanente en memoria principal, como el kernel y se puede copiar a disco como cualquier programa ordinario. El sistema permite que los usuarios realicen modificaciones sobre él.

En general, el shell proporciona un mejor ambiente para ejecutar programas que involucran manipulación de archivos o textos y manipulación numérica o de computación. En este último caso, un lenguaje de programación convencional podría proporcionar una mayor rapidez y flexibilidad.

En este sentido estricto, es un procesador de comandos, pero funcionalmente es un lenguaje de programación. Como tal, le es posible crear programas completos de aplicación ayudado de algunas utilerías.

III.1. SUSTITUCIÓN DE METACARACTERES

Shell reconoce especialmente a algunos caracteres, a estos se les reconoce como "metacaracteres" (wildcards). La siguiente tabla muestra los metacaracteres utilizados por el shell y sus propiedades especiales.

Metacaracter	Propiedades
>	*prog* > *file* dirigir la salida estándar hacia *file*.
>>	*prog* >> *file* agregar la salida estándar a *file*.
<	*prog* < *file* extraer de *file* la entrada estándar.
|	P1 | p2 conectar la salida estándar de p1 a la entrada estándar de p2.

<<str	sigue la entrada estándar hasta la siguiente *str* en el renglón
*	concordar cualquier cadena de cero o de más caracteres en archivos.
?	concordar cualquier carácter individual e archivos.
[ccc]	concordar cualquier carácter individual de ccc en archivos, son legales los intervalos como 0-9 ó a-z.
;	terminador de comando: p1;p2 hace p1, entonces p2
&	como ; pero no espera que termine p1
`...`	correr comandos(s) en ...; la salida sustituye a `...`
(...)	correr comandos(s) en ... en un subshell
{...}	correr comandos(s) en ... shell actual (de poco uso)
$1, $2, etc	$0... $9 reemplazada por los argumentos en el archivo de shell.
$var	valor de la variable de shell var
${var}	valor de var; evita confusión cuando está concatenada con el texto
\	\c tomar literalmente el carácter c,\newline desechado
'...'	tomar ... literalmente
"..."	tomar ... literalmente después de $ `...` y \ interpretando
#	si # comienza una palabra, el resto de la línea es un comentario (ausente en la séptima edición)
var=value	asignar a la variable var
p1 && p2	ejecutar p1; si se logra, ejecutar p2
p1 || p2	ejecutar p1; si no se logra, ejecutar p2

III.2. SUBSTITUCIÓN DE CARACTERES

* Indica cualquier cadena de cero o más caracteres exceptuando la diagonal, /.

? Indica un sólo carácter individual en una cadena.

[.. Los caracteres que se encuentran dentro de los corchetes especifican el conjunto de caracteres que se van a buscar.
Son legales los intervalos 0-9 o a-z.

\\c Tomar literalmente el carácter c.

`...` Tomar...literalmente.

"..." Tomar literalmente.

Todo lo que siga de # se anula.

^ Para indicar el inicio de la línea.

$ Para indicar el final de la línea.

III.2.1. VARIABLES

S 0...S9 Reemplaza por los argumentos en el archivo de shell
S var Valor de la variable de shell var
S(var) Valor de var: evita confusión cuando está conectada con el texto.
v=val Asigna un valor a la variable v

III.3. COMUNICACIÓN ENTRE PROCESOS

> prog > tmp	dirigir la salida estándar hacia el archivo tmp.
>> prog >> tmp	agregar la salida estándar al archivo tmp.
< prog < tmp	extraer del archivo tmp la entrada estándar.
| p1 | p2	conectar la salida estándar del p1 a la entrada estándar de p2.

III.3.1. PROCESOS

; Terminador de comando: p1;p2 hace p1 y después p2
& Se ejecuta el proceso en *bakground*. ejemplo:

procesc &p1&p2 Ejecuta p1: si se logra. ejecuta p2
p1|| Ejecuta p1: si no se logra ejecuta p2

Para anular el efecto de los caracteres especiales se puede encerrar entre apóstrofos o entre comillas o poner una diagonal invertida antes de cada carácter.

echo '***'
echo ***
$ rm* **CUIDADO AL UTILIZAR ESTE COMANDO, BORRA TODO EL CONTENIDO DEL DIRECTORIO.**

$ ECHO a'*'b
a*b
4 echo '*'A'?'
*A?
$ echo a* b **despliega todos los nombres de archivos que comiencen con A y terminen con B.**

$ echo `who | wc '1' DESPLIEGA EL NÚMERO DE USUARIOS.

$ grep juan < telefonos
$ banner 'date'

III.4. ABREVIATURAS DE NOMBRES DE ARCHIVOS

Comencemos con los patrones de nombres de archivos. Supóngase que se está tecleando un documento grande, digamos un libro. Lógicamente éste se divide en partes de menor tamaño, tales como capítulos y tal vez secciones. También físicamente habrá de estar dividido, debido a la incomodidad que representa la edición de archivos extensos. Así, el documento debe ser tecleado en varios archivos. Se podría tener archivos separados para cada capítulo, llamados ch1,ch2 etc.

chi.1

ch1.2

ch1.3

. . .

ch2.1

ch2.2

. . .

Con una nomenclatura sistemática, se puede saber rápidamente dónde encaja un archivo dentro del total.

¿. Y si el usuario quisiera imprimir todo el libro ? Podría teclear

S pr **ch1.1 ch1.2 ch1.3** . . .

pero pronto se aburriría de teclear tantos nombres de archivos y empezaría a cometer errores. Aquí es donde entran en acción las abreviaturas de nombres de archivos. Si se teclea

$ pr ch*

el shell toma el * como " cualquier cadena de caracteres ", de modo que ch* es un patrón que corresponde a todos los nombres de archivos en el directorio actual que empiecen con ch. El shell crea la lista, en orden alfabético y la pasa a pr. El comando pr nueva ve el *. La búsqueda que hacer el shell en el directorio actual genera una lista de cadenas que se pasan a pr.

El punto crucial es un servicio del shell. Por ello puede utilizarse para generar una secuencia de nombres de archivos destinados a cualquier comando. Por ejemplo, para contar las palabras en el primer capítulo:

$ wc ch1.*

113 562 3200 ch1.0

935 4081 22435 ch1.1

974 4191 22756 ch1.2

378 1561 8481 ch1.3

1293 5298 28841 ch1.4

33 194 1190 ch1.5

75 323 2030 ch1.6

3801 16210 88933 total

Existe un programa llamado **echo que es especialmente** útil para ensayar el significado de los caracteres de **abreviatura**. Como puede suponerse, echo no hace otra cosa que retransmitir (" **hacer eco a** ") sus **argumentos** a la terminal:

$ echo hello world

hello world

$

Pero los argumentos pueden ser **generados por medio de** patrones:

$ echo ch1. *

lista los nombres de todos los archivos **del** capítulo 1,

$ echo *

lista por orden alfabético todos los nombres de archivos que se encuentren en el directorio de trabajo,

$ pr *

imprime todos los archivos del usuario (en orden alfabético), y borra todos los archivos en el directorio de trabajo. (Se debe estar muy seguro de que esto es lo que desea hacer).

$ rm *

El * no está limitado a la última posición en un nombre de archivo. Se pueden colocar uno o varios de ellos en cualquier posición del nombre de archivo. Así,

$ rm *.save

elimina todos los archivos que terminan con .save.

Obsérvese que los nombres de archivos se ordenan alfabéticamente, lo cual no es lo mismo que si estuvieran ordenados numéricamente. Si el libro tiene 10 capítulos, el orden tal vez no sea el deseado por el usuario, ya que ch queda antes que ch2:

$ echo *
ch1.1 ch1.2 . . . ch10.1 ch10.2 . . . ch2.1 ch2.2 . . .
$

El * no es la única característica de reconocimiento de patrones del shell, aun cuando es el más frecuentemente usado. El patrón [...] representa cualquiera de los caracteres dentro de los corchetes. Podemos abreviar un rango de letras o dígitos consecutivos:

$ pr ch [12346789)*	*Imprimir los capítulos, 1,2,3,4,6,7,8,9 pero no el 5*
$ pr ch [1-46-9)*	*Los mismos que antes*
$ rm temp [a-z)	*Elimine los tempa, ...tempz que existan*

El patrón ? presenta a cualquier carácter:

$ ls ?	*Mostrar los archivos con nombre de un solo carácter*
$ ls -1 ch? .1	*Mostrar ch1.1 ch2.1 ch3.1 etc., pero no ch 10.1*

$ rm temp? *Eliminar los archivos temp1,... tempa, etc.*

Nótese que estos patrones funcionan sólo en nombres de archivos existentes. En particular, no es posible crear nuevos archivos usando patrones. Por ejemplo, si se desea expandir ch chapter en cada nombre de archivo, eso no puede hacerse de la siguiente manera:

$ mv ch.* chapter.* *¡ No funciona!*

ya que chapter.* no concuerda con ningún archivo existente.

Los caracteres de patrones tales como * pueden usarse tanto en trayectorias como en nombres de archivos simples; la búsqueda se hace para cada componente de la trayectoria que contenga un carácter especial. Así, /usr/mary/ * lleva a cabo la búsqueda dentro de / usr / mary, y / usr / * / calendar genera una lista de nombres de todos los archivos calendar del usuario.

Si se desea alguna vez eliminar el significado especial de *, ?, etc., deberá encerrarse el argumento completo entre apóstrofos como en

$ ls ' ? '

Esto también puede hacerse anteponiendo al carácter especial una diagonal invertida:

$ ls \?

(Recuérdese que como ? no es el carácter de borrado o cancelación de líneas, esa diagonal invertida es interpretada por el shell y no por el núcleo).

La mayor parte de los comandos que se han visto hasta aquí producen su salida en la terminal; algunos otros, tales como el editor, también toman su entrada de la terminal. Es una regla casi universal el que la terminal pueda ser reemplazada por un archivo para la entrada, la salida o ambas. He aquí un ejemplo:

$ ls

que genera una lista de nombres de archivo en la terminal del usuario. Pero si se teclea

$ ls >filelist

la misma lista de nombres de archivo será colocada en el archivo filelist. El símbolo > , significa "coloca la salida en el siguiente archivo, en vez de en la terminal". El archivo será creado en caso de que no existiera ya anteriormente y, si ya existía, su contenido será reemplazado. Nada se escribirá en la terminal. He aquí otro ejemplo. Es posible cambiar varios archivos en uno capturando la salida de cat dentro de un archivo:

$ cat f1 f2 f3 >temp

El símbolo >> opera al igual que >, excepto que aquél significa "anexa al final de". Esto es,

$ cat f1 f2 f3 >>temp

copia el contenido de f1, f2 y f3 al final de lo que se encuentra en temp, en vez de reemplazar el contenido existente. Como en >, si temp no existe, se creará inicialmente vacío para el usuario.

En una forma similar, el símbolo > significa tomar la entrada de un programa desde el archivo siguiente en vez de tomarlo de la terminal. Así, se puede preparar una carta en el archivo let, para después enviarla a varias personas con

$ mail mary joe tom bob <let

En todos estos ejemplos, los espacios en blancos son opcionales en ambos lados de >o<, si bien nuestro formado es tradicional.

Dada la capacidad de redireccionar la salida mediante >, ahora es factible cambiar comandos para lograr efectos que no hubieran sido posibles de otra manera. Por ejemplo, para desplegar una lista alfabética de usuarios,

$ who >temp

$ sort <temp

Puesto que who despliega una línea de salida por cada usuario que se encuentre en sesión, y wc -l cuenta con líneas (suprimiendo el conteo de palabras y caracteres), se puede encontrar el número de usuarios con

$ who' >temp

$ wc -1 <temp

También se puede contar los archivos en el directorio de trabajo mediante

$ ls >temp

$ wc -1 <temp

aunque esto incluirá también temp en el conteo. Para imprimir los archivos en 3 columnas se tecleará

" ls >temp

$ pr -3 <temp

Y se puede saber si un usuario se encuentra en sesión combinando who y grep:

$ who >temp

$ grep mary <temp

En todos estos ejemplos, al igual que en los caracteres de patrones para nombres de archivos, tales como *, es importante recordar que la interpretación de > y < la hace el shell y los programas individuales. Centralizar esta facilidad

en el shell significa que el redireccionamiento de entrada y salida puede usarse con cualquier programa; el programa mismo no sabe que ha ocurrido algo fuera de lo común.

Esto nos lleva a una importante convención . El comando

S sort <temp

Ordena los contenidos del archivo temp al igual que

S sort temp

Pero existe una diferencia. Como la cadena < temp es interpretada por el shell, sort no ve como argumento el nombre de archivo temp; sino que ordena su *entrada estándar* , que el shell ha redireccionado para que provenga del archivo. Sin embargo el último ejemplo pasa el nombre temp como argumento para sort, el cual lee el archivo y lo ordena. Es posible pasar a sort también una lista de nombres, como en

S sort temp1 temp2 temp3

Pero si no se le proporciona ningún nombre de archivo, ordena su entrada estándar. Esto es una propiedad esencial de la mayoría de los comandos; si no se especifican nombres de archivos, se procesa la entrada estándar. Esto significa que el usuario puede simplemente teclear comandos para ver como funciona. Por ejemplo,

S sort

ghi

abc

III.5. INTERCONEXIONES (PIPES)

Una interconexión *(pipe)* es una forma de conectar la salida de un programa a la entrada de otro programa sin ningún archivo temporal; se puede establecer interconexiones entre dos o más programas a través de esto.

El carácter de barra vertical | indica al shell que establezca una conexión:

$ who | sort

$ who | wc -l

$ ls | wc -l

$ ls | pr -3

$ who | grep mary

Imprimir lista clasificada de usuarios

Cuente los usuarios

Cuente los archivos

Lista a tres columnas de los archivos

Busque un usuario en particular

Cualquier programa que lee desde una terminal también puede leer de una interconexión, cualquier programa que escriba en una terminal puede, asimismo, escribir a una interconexión. Aquí es donde interviene la convención de leer de la entrada estándar cuando no se hace referencia a ningún archivo: cualquier programa que siga esta convención puede usarse en interconexiones. grep, pr, sort, y wc.

En una interconexión se puede tener todos los programas que se deseen.

$ ls | pr -3 | lpr

crea una lista de archivos a 3 columnas en la impresora, y

$ who | grep mary | wc -l

Los programas en una interconexión en realidad corren el mismo tiempo, no uno después del otro. Esto significa que esos programas pueden ser interactivos; el núcleo se encarga de cualquier arreglo y sincronización que sean necesarios para que esto funcione.

El shell se encarga de las cosas cuando invoca a una interconexión; los programas individuales no se ocupan del redireccionamiento. Por su puesto, los programas tienen que operar debidamente para ser combinados de esta manera. La mayor parte de los comandos siguen un diseño común, por lo que encajarán

correctamente en las interconexiones en cualquier posición. Por lo regular una invocación de un comando es como sigue

comando argumentos-opcionales nombres-de-archivos-opcionales

Si no se dan nombres de archivos, el comando lee de su entrada estándar, que por default es la terminal (lo resulta cómodo para ensayar los comandos), pero ésta puede ser redireccionada para que provenga de un archivo o una conexión. Por otra parte, en lo que respecta a la salida, la mayoría de los comandos escriben su *salida estándar*, que por default se envía a la terminal. Pero también puede ser redireccionada hacia un archivo o una interconexión.

Los mensajes de error de los comandos deben ser manejados en forma diferente; de otro modo desaparecerían en un archivo o en una interconexión. Así, cada comando posee también una salida de *error estándar* , la cual está dirigida normalmente a la terminal.

La mayoría de los comandos de los que hemos hablado hasta aquí encajan en este modelo; las únicas excepciones son comandos como date y who, que no leen ninguna entrada, y algunos como cmp y diff, que poseen un número determinado de archivos de entrada.

S who | sort

S who >sort

III.6. PROCESOS

El shell lleva a cabo algunas cosas además de activar interconexiones. Veamos brevemente algo acerca de los aspectos básicos de la ejecución de más de un programa a la vez, puesto que ya hemos visto algo de eso al hablar de las interconexiones. Por ejemplo, se pueden ejecutar dos programas en una sola línea de comandos separando los comandos con un punto y coma; el shell reconoce el punto y coma y parte la línea en dos comandos:

```
S date; who
True Sep 27 01:03:17 EDT 1983
ken     ttyo    Sep 27 00:43
drm     tty1    Sep 26 23:45
rob     tty2    Sep 26 23:59
bwk     tty3    Sep 27 00:06
jj      tty4    Sep 26 23:31
you     tty5    Sep 26 23:04
ber     tty7    Sep  26 23:34
S
```

Ambos comandos son ejecutados (en secuencia) antes de que el shell regrese con el carácter de espera.

También es posible tener más de un programa en ejecución simultáneamente si se desea. Por ejemplo, supóngase que se desea hacer algo que consume tiempo, como contar el número de palabras en un libro, pero no se desea esperar a que termine el comando wc para empezar a hacer otra cosa. Entonces puede teclearse.

$ wc ch* > wc.out &

6944 *Identificador del proceso impreso por shell*

El signo & al final de una línea de comandos indica al shell "empieza a ejecutar este, comando; y pasa inmediatamente a ejecutar los siguientes comandos de la terminal", esto es, sin esperar a que termine el primero. De este modo, el comando comenzará a ejecutarse pero mientras el usuario puede hacer alguna otra cosa. El direccionamiento de la salida hacia el archivo wc. out evita que el comando interfiera con lo que está haciendo en ese momento.

Un programa en ejecución se llama *proceso.* El número desplegado por el shell cuando un comando fue lanzado con & se llama *identificador del proceso;* el usuario puede usarlo en otro comandos para referirse a un programa específico en ejecución.

Es importante distinguir entre programas y procesos. wc es un programa; cada

vez que se ejecuta el programa wc, se crea un nuevo proceso. Si varios de estos programas se ejecutan al mismo tiempo, cada uno de ellos es un programa específico en ejecución.

Si una interconexión se inicia con &, como en

$ pr ch* | lpr &

6951 *Identificador de proceso de lpr*

los procesos en ella empezarán todos de inmediato; el signo & se aplica a la interconexión. De cualquier modo sólo se despliega un identificador de proceso, para el último de la secuencia.

El comando **wait** espera hasta que todos los procesos lanzados con & terminen. Si no regresa inmediatamente, es que se tienen aún comandos en ejecución. Puede interrumpiste el wait por medio de DELETE. El usuario puede utilizar el identificador de proceso desplegado por el shell para detener un proceso iniciado con &:

$ kill 6944

Si el lector olvida el identificador de proceso, puede usar el comando ps para obtener información de lo que se está ejecutando. Si está desesperado, kill 0 descontinuará todos los procesos excepto su shell de inicio de sesión. Y si quiere saber lo que otros usuarios están haciendo, ps-ag desplegará información acerca de todos los procesos que se están ejecutando en ese momento. A continuación se muestra un ejemplo de la salida de ps:

```
4 ps -ag
  PID  TTY TIME CMD
   36  co  6:29/etc/cron
 6423  5   0:02 -sh
 6704  1   0.04 -sh
 6722  1   0.12 vi paper
 4430  2   0:03 -sh
 6612  7   0:03 -sh
 6628  7   1:13 rogue
 6843  2   0:02 write dmr
 6949  4   0:01 login bimmler
 6952  5   0:08 pr ch1.1 ch1.2 ch1.3 ch1.4
 6951  5   0:03 lpr
 6959  5   0:02 ps -ag
 6844  1   0:02 write rob
S
```

PID es el process-id (identificador del proceso);TTY la terminal asociada con el proceso (como en who);TIME es el tiempo de procesador utilizado, en minutos y segundo; y el resto es el comando en ejecución, ps es uno de los comandos que difieren en versiones distintas del sistema.

Los procesos poseen la misma clase de estructura jerárquica que los archivos: cada proceso tiene un padre y puede también tener hijos. El shell del usuario fue creado por un proceso asociado con la terminal que lo conecta al sistema. mientras se ejecutan comandos, esos procesos son los hijos directos del shell.

Si se ejecuta un programa desde uno de ellos, por ejemplo con el comando ! para salir temporalmente de ed, él (de) crea su propio proceso hijo, el cual es entonces nieto del shell.

Algunas veces un proceso tarda tanto tiempo en ejecutarse que sería deseable que empezara a correr y uno pudiera apagar la terminal sin esperar a que termine. Pero si el usuario la apaga o interrumpe su conexión, el proceso por lo general será descontinuado aun si se utilizó **&**. El comando nohup ("no hangup": "sin cortar") fue creado para resolver esta situación: si se teclea

$ nohup *comando* **&**

El comando continuará ejecutándose aún si uno termina su sesión. Cualquier salida del comando se guardará en un archivo llamado nohup.out. No existe forma de ejecutar un comando con nohup de modo retroactivo.

Si un proceso requiere muchos recursos del procesador, es conveniente para los que comparten el sistema con el usuario que éste realice su tarea con prioridad menor que la normal; esto se hace con otro programa llamado nice:

$ nice *comando-costoso* **&**

nohup automáticamente llama a nice, ya que si el usuario va a terminar la sesión puede permitir que el comando tarde un poco más en su ejecución. Finalmente, el lector puede indicar al sistema que empiece a ejecutar su proceso a alguna hora de la madrugada cuando la gente normal duerme. El comando se llama at (1):

$ at *hora*

cualquier comando

que desee...

***ctl*-d**

$

Es la manera tipica de usarlo, aunque los comandos también podrían estar en un archivo:

$ at 3am <file

$

Las horas pueden escribirse tipo 24 horas, como 2130, o tipo 12 horas, como 930 pm.

III.7. ADAPTACIÓN AL AMBIENTE

Una de las virtudes del sistema UNIX es que existen varias maneras de adecuarlo al gusto del usuario o a las convenciones de su entorno local de computación. Por ejemplo, hemos mencionado anteriormente el problema de los distintos estándares para los caracteres de borrado y cancelación de líneas, que por default son generalmente #. Pueden cambiarse cuando se desee por medio de

$ stty erase *e* kill *k*

donde *e* es cualquier carácter que se desee para borrar y *k* para cancelar líneas. Sin embargo, sería engorroso tener que teclear esto cada vez que se inicie una sesión.

El shell soluciona el problema. Si hay un archivo llamado *profile* en el directorio de inicio de sesión, el shell ejecutará los comandos contenidos en él

cuando se entre en sesión, antes de desplegar el símbolo de espera por primera vez. Uno puede por tanto poner comandos en · profile para cambiar el ambiente a la forma deseada, y éstos serán ejecutados cada vez que se inicie una nueva sesión. Lo primero que la mayoría de la gente pone en su. profile es

stty erase -

Hemos usado aqui el carácter- para que sea visible, pero podria ponerse un carácter de retroceso en forma literal en el archivo · profile.sttty también entiende la notación ^x para ctl-x, por lo que se obtendría el mismo resultado con

stty erase '^h'

Ya que ctl-h es el carácter de retroceso. El carácter ^ es un sinónimo obsoleto del operador de interconexión (pipe):, por lo que debe ser delimitado con apóstrofos.

stty erase '^h' -tabs

Si se desea conocer qué tan ocupado está el sistema cuando se entra en sesión, agregese:

who | wc -l

para contara los usuarios. Si existe un servicio de noticias en el sistema, se podria añadir news. Algunas personas desearán una predicción impresa:

/usr/games/fortune

Al cabo de un rato quizá el usuario decida que el inicio de sesión se está tardando demasiado y deduzca ·profile al mínimo indispensable.

Algunas de las propiedades del shell son propiamente controladas por las *variables de shell* con valores que el usuario puede accesar y asignar. Por ejemplo, el símbolo de espera, que hemos mostrado hasta aquí como **$** , está en realidad almacenado en una variable de shell llamada ps1, y se le puede asignar lo que uno desee; por ejemplo

ps1= ''¿A sus ordenes?''

Los apóstrofos son necesarios puesto que existen espacios en blanco en la cadena. Los blancos no se permiten alrededor del signo = en esta construcción.

El shell también trata en forma especial las variables HOME y MAIL. HOME es el nombre del directorio de origen del usuario; normalmente se asigna sin que esté en profile. La variable MAIL contiene el nombre del archivo estándar donde se almacena el correo del usuario. Si se define para el shell, se notificará al usuario si ha llegado más correo de cada comando:

MAIL= /usr/spool/mail/you

(El archivo de correo puede ser diferente en el sistema del lector, a menudo se usa también /usr /mail /you).

Probablemente la variable de shell más utilizada es la que controla dónde busca los comandos el shell. Recuérdese que, cuando se teclea el nombre de un comando, el shell casi siempre lo busca primero en el directorio actual, después en /bin y luego en /usr/bin. Esta secuencia de directorios se llama *ruta de búsqueda* y se almacena en la variable de shell llamada PATH. Si la ruta de búsqueda normal no es lo que el usuario desea, se puede cambiar en el archivo profile. Por ejemplo, la siguiente línea asigna a la ruta lo estándar más un directorio adicional, que es /usr/games:

PATH = .:/bin:/usr/bin:/usr/games *Una forma...*

La sintaxis parece extraña: una secuencia de nombres de directorios separados por dos puntos. Recuérdese que '.' es el directorio de trabajo. Este puede omitirse; un componente nulo en PATH representa el directorio de trabajo. Otra forma de hacer lo anterior es simplemente aumentar el valor anterior:

PATH= $PATH:/usr/games *... Otra forma*

Se puede obtener el valor de cualquier variable de shell anteponiendo a su nombre un **$**. En el ejemplo anterior, la expresión $ PATH obtiene el valor

actual, al cual se añade la nueva parte, y el resultado se asigna otra vez a PATH . Lo que se hizo se puede verificar con echo:

S echoPATH is SPATH

PATH is :/bin:/usr/bin:/usr/games

S echo SHOME *Su directorio de inicio de sesión*

/usr/you

Si el usuario tiene algunos comandos propios, éstos se pueden reunir en el directorio y añadirlo a la ruta de búsqueda. En ese caso, la asignación se haría en la siguiente forma:

PATH= SHOME/bin:/bin:/usr/bin:/usr/games

Otra variable, usada frecuentemente por editores de texto más refinados que de, es TERM, la cual contiene el tipo de terminal que se está usando. Dicha información puede permitir que los programas manipulen la pantalla en forma más eficiente. Así, se podría agregar al archivo profile algo como esto:

TERM=adm3

También es posible emplear variables para hacer abreviaturas. Si uno se refiere continuamente a algún directorio en particular con nombre muy extenso, valdría la pena añadir al archivo profile una línea como la siguiente:

d= \|*directorio*\| *horriblemente* \|*largo* \|*nombre*

de manera que pueda teclearse después

$ cd $d

Las variables personales como d se escriben por convención en minúsculas para distinguirlas de las utilizadas por el shell, como PATH. Finalmente, es necesario indicar al shell que uno quiere usar las variables también en otros programas; esto se hace por medio del comando export.

export MAIL PATH TERM

CAPITULO IV. UTILERIAS Y APLICACIONES.

Objetivo:

IDENTIFICAR LAS PRINCIPALES UTILERIAS DE MANEJO DE VENTANAS QUE PROPORCIONA UNIX, ASÍ COMO LA MANERA DE UTILIZARLAS.

CAPÍTULO IV. UTILERIAS Y APLICACIONES.

El sistema UNIX ofrece muchas más aplicaciones más características que las ya mencionadas.

IV.I. INTRODUCCIÓN AL USO DE VENTANAS

En todas las versiones de UNIX se trabaja en terminales de computadora de una sola ventana, en una sesión única de trabajo. En las versiones más recientes de UNIX hay uno o varios sistemas de ventanas. Un sistema de ventanas es un paquete de programas mediante el cual una terminal lleva acabo varias sesiones al mismo tiempo. Además del teclado, en los sistemas de ventanas se utiliza el ratón o algún otro dispositivo, como la bola redonda (trackball), para mover el apuntador (pointer) por la pantalla. Este apuntador sirve para seleccionar porciones de la pantalla y desplazarlas, copiarlas o pegarlas, para trabajar con menús de comandos y para muchas cosas más. Si alguna vez ha trabajado en una Macintosh, con Microsoft Windows, con el OS/2 y su Presentation Manager (Administrador de presentaciones), entre otros sistemas, entonces usted ya ha trabajado con un sistema de ventanas. En la figura 4.1. se muestra la pantalla característica de un sistema de ventanas.

En este capitulo se presenta el Xwindow (para ser breves, lo llamaremos X), el sistema de ventanas más común de UNIX. Esta presentación también le servirá

para aprender a manejar otros sistemas que no están basados en Xwindow.

Al igual que UNIX, X es muy flexible. La aparición de las ventanas, la forma de operar de los menús y otras funciones más se controlan mediante un programa, conocido como administrador de ventanas (Windows Manager).

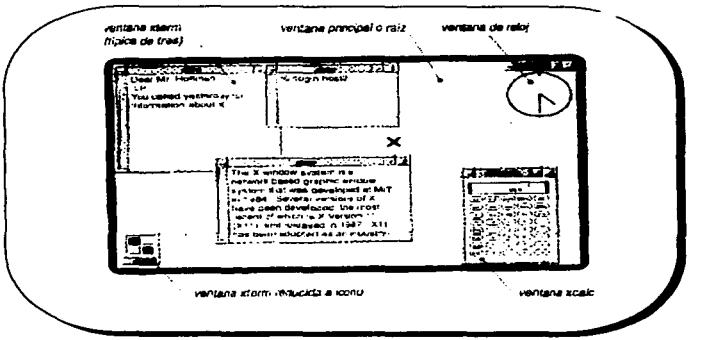

figura 4.1 LA pantalla X y el administrador de ventanas mwm.

Tres de los administradores más comunes son **mwm** (Motif Window Manager o Administrador de ventanas de motivos), **olwm** (Open Look Window Manager o Administrador de ventanas de visión abierta) y **twm** (Tab Window Manager o Administrador de ventanas de Tabuladores). Hay muchos otros administradores de ventanas así como en la forma en que aparece en la pantalla.

IV.2. CÓMO INICIAR EL PROGRAMA X.

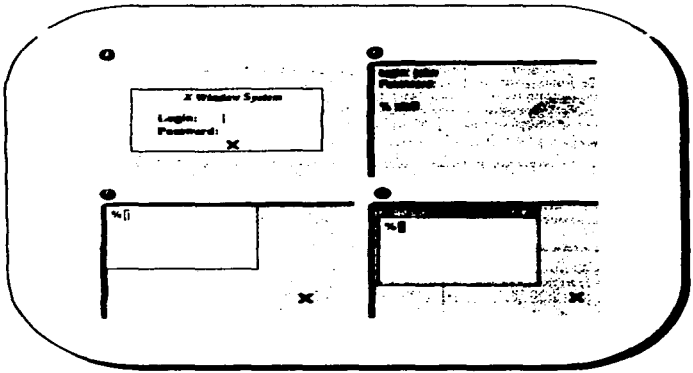

figura 4.2 se muestran ciertos procedimientos y rutas para iniciar el X.

- En la figura 4,2A, **xdm** está activo.
- En la figura 4.2B, hay una sesión de UNIX login (de entrada) estándar.
- En la figura 4.2C, X está activo aunque es probable que el administrador de ventanas no lo esté. (Lo reconocerá porque la ventana no tiene marco).
- En la figura 4.2D, la ventana está enmarcada; quiere decir que X y el administrador de ventanas **mwm** están activos.

IV.2.1. Ejecución de x (con xdm).

Algunas terminales, como en el caso mostrado en la figura 4.2A, ya están listas para el X. Después de iniciar, aparecerá una sola ventana en medio de la ventana, con una leyenda como " X Window System sobre hostname (el nombre del servidor) ". La ventana presenta dos indicadores (prompts) adicionales: " login " (entrada) " password ": (contraseña). A la derecha de la línea (login) puede observarse un cursor (la barra vertical) estático. teclee su nombre de usuario (username, login name) y pulse **ENTER** . Haga lo mismo para la contraseña. Desaparecerá la ventana de login aparecerá una pantalla semejante a la de la figura 4.1.

¡Ya podrá usar X! es probable que su terminal este configurada para funcionar con **xdm,** el administrador de pantalla X y; **xdm** le permite conectarse a su servidor UNIX y, por lo general, se inicia el administrador de ventanas. Ahora, puede pasar a la siguiente sección: " Ejecución de programas "

IV.2.2. Inicio del x en una sesión estándar de UNIX.

Si en su terminal apareciera algo semejante a lo de la figura 4.2B, con un indicador de UNIX login: estándar (pero no en una ventana aparte, y toda la pantalla pareciera como una terminal), querrá decir que X no está funcionando. Inicie la sesión hasta que aparezca el indicador del shell (por ejemplo. %) luego,

tendrá que iniciar X. El comando por omisión es:

% xinit

IV.2.3. El administrador de ventanas.

Una vez abierta la ventana, en la que debe aparecer el indicador del shell (por lo general % o $), ya se puede iniciar el programa administrador de ventanas. Es posible que su cuenta este configurada para realizar lo anterior de manera automática. Si el administrador de ventanas no está operando, las ventanas aparecerán sin marcos (con titulos, cuadros de control, etc.). Por otra parte, cuando usted lleve el apuntador (puntero) a la ventana raíz o principal (también conocida como " escritorio ") y pulse los botones del ratón, los menús no aparecerán a menos que el administrador de ventanas ya este operando.

Si quiere iniciar el administrador de ventanas, ponga el apuntador dentro de la ventana. Introduzca el siguiente comando en el indicador (prompt) del shell:

% mwm &

(1) 12345

%

(Para iniciar el **twm**, y el **olwm** u otro administrador de ventanas, como comando debe teclear de ese administrador de ventanas). Poco después, la

ventana tendrá ya un marco.

IV.3. EJECUCIÓN DE PROGRAMAS.

Una de las características más importantes de X es que las ventanas pueden originarse ya sea en programas que se estén ejecutando en otra computadora o en un sistema operativo que no sea UNIX. Es decir, si su programa MS-DOS favorito no se puede ejecutar en UNIX pero tiene interfaz X, lo único que habrá que hacer es ejecutarlo en MS-DOS y luego desplegar sus ventanas con X en su computadora con UNIX. Por ejemplo, esto permite que un investigador ejecute programas de análisis gráficos de datos en supercomputadoras de un sitio determinado al tiempo que podrá ver los resultados en su oficina.

IV.3.1. Para fijar el foco de entrada.

De todas las ventanas que aparecen en pantalla, solo una de ellas recibe lo que usted escribe con el tecleado. Por lo general esta ventana se resalta de alguna manera. Por ejemplo, en el caso del administrador de ventanas **mwm** , por omisión el marco de la ventana en que se recibe lo que se escribe es de un gris más obscuro. En la jerga de X, la elección de la ventana en donde se recibe lo que se escribe se denomina " fijar el foco de entrada " . La gran mayoría de los administradores de ventanas se configuran para que la fijación del foco se realice de alguna de las dos formas siguientes:

- Apunte a la ventana deseada y haga clic con el botón del ratón (por lo general, el izquierdo). Posiblemente también tenga que hacer clic en la barra de título, que está en la parte superior de la ventana.
- Sólo ponga el apuntador dentro de la ventana.

Cuando usted utiliza **mwm**, en toda ventana nueva aparecerá automáticamente el foco de entrada .

IV.3.2. Cómo trabajar con el ratón.

Ahora, veremos algunos procedimientos básicos para manejar el ratón y otros dispositivos de señalamiento.

La forma del apuntador.

Conforme se desplaza el apuntador* (o puntero) del ratón de la ventana principal a otras ventanas o menús, la forma del apuntador cambia. Por ejemplo, cuando está en la ventana principal, el apuntador aparecerá como una **X** grande. Esta forma se transforma en la de un reloj de arena para indicarle que debe esperar. Al cambiar el tamaño de una ventana, el apuntador cambia a la forma de una cruz con flechas.

Cómo apuntar, hacer clic y arrastrar.

¿Qué es apuntar y hacer clic? es trasladar el apuntador al un sitio (por lo general, a alguna parte de la ventana), luego oprimir y liberar rápidamente el botón del ratón (casi siempre, el botón izquierdo). Es lo mismo que cuando se oprime un botón del aparato telefónico u otro aparato.

También se utiliza el "arrastre". Es decir, mover el apuntador a un sitio (por ejemplo, la esquina de una ventana), sostener el botón del ratón y mantenerlo así al tiempo que se mueve el apuntador. A esto de denomina "arrastrar" un apuntador u objeto, debido a que el objeto se va arrastrando junto con el apuntador hasta que se deja de pulsar el botón del ratón.

Como usar el ratón con ventanas xterm.

Las ventanas de **xterm** tiene la ventaja, sobre las terminales de UNIX sencillas de que usted puede copiar y pegar textos dentro de una ventana o entre varias ventanas. Para empezar, ponga el apuntador dentro de una ventana **xterm** y escoja la ventana (fije en ella el foco). Observará que el apuntador adquiere entonces la forma de "Y" . También aparecerá un cursor en forma de bloque. Observe cómo al ir escribiendo el texto, éste aparece en el cursor en forma de bloque tal como sucedería en una terminal estándar; es decir considere el cursor de bloque como el punto de entrada de la ventana.

El apuntador en forma de **I** selecciona el texto que se va a copiar. Probemos. Apunte al primer carácter de una línea de comando (no el indicador) y haga clic en el botón izquierdo del ratón. Luego, desplace el apuntador a final del texto que desee seleccionar y haga clic en el botón derecho. El texto comprendido entre los dos clic deberá aparecer resaltado. (si se equivoca al hacer clic, tendrá que repetir todo el procedimiento). Su ventana **xterm** deberá parecerse a la de la figura 4.3 .

Después, haga clic (no pulse continuamente) con el segundo botón del ratón (el de en medio). El texto seleccionado se copiará en la ventana en la que se encuentra el cursor de bloque como si usted hubiera tecleado en él. Oprima **ENTER** para que se ejecute la línea de comando; de no hacerlo así retroceda hasta llegar al indicador.

Para seleccionar texto también se puede hacer clic en una ventana **xterm.** Apunte a una palabra y haga doble clic (dos clics, con rapidez) con el botón izquierdo. La palabra deberá aparecer resaltada. Apunte a una línea y haga triple clic para resaltarla. Luego, puede seleccionar y copiar cualquier texto, no sólo líneas de comandos.

El copiado y pegado funciona también en las ventanas **xterm** y en otras ventanas (aunque no en todas) que manejan texto. Puede seleccionar texto en una ventana y pegarlo en otra. Esto resulta muy práctico en las tareas de edición

de texto.

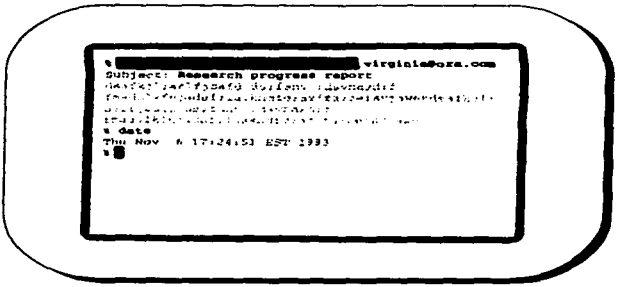

figura 4.3: Copiado y pegado de una línea de comandos.

Cómo trabajar con ventanas.

El administrador de ventanas le ayudará a controlar las ventanas. En está sección se le explicará cómo maneja **mwm** las ventanas. Los demás administradores hacen lo mismo, aunque con algunas variantes, empecemos por examinar la figura 4.4, una ventana típica controlada por **mwm** .

En la parte superior está la barra de título, con el título de la ventana y tres botones. Los extremos de la ventana sirven para el manejo del tamaño.

Cómo usar la barra de títulos.

En la parte superior de la ventana hay tres botones (Véase la figura 4.4). En el interior de los dos botones del extremo superior derecho hay sendos cuadros. Al hacer clic en el botón del cuadro pequeño, la ventana se reduce a su mínima dimensión; la ventana se convierte en un icono. La reducción al tamaño de icono de las ventanas que no se necesitan permite abrir espacios sin suprimir los programas de cada una de las ventanas reducidas; además, eso impide que usted trabaje en una ventana indebida. (En la figura 4.1 se muestra un icono). El botón del cuadro grande sirve para maximizar la ventana. Permite ampliarla todo lo que el cliente acepte, aveces ocupando toda la pantalla.

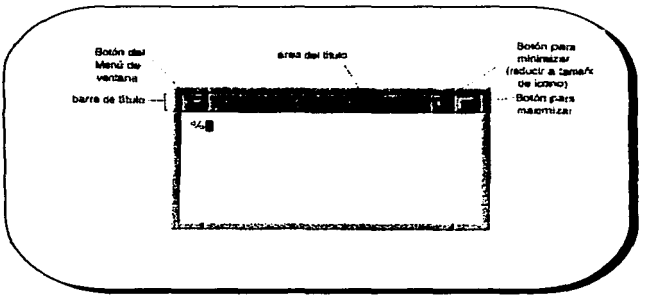

figura 4.4. Una ventana que opera con el Administrador de ventanas Motif.

El botón de la izquierda sirve para abrir el menú de la ventana.

Como trasladar ventanas e íconos.

Para dezplazar una ventana empiece por señalar la barra de títulos. Para mover un icono, señálelo. Mantenga pulsado el botón izquierdo del ratón y arrastre a la nueva ubicación; luego, suelte el botón. También se puede iniciar el desplazamiento desde el menú de la ventana, aunque consideramos que el primer procedimiento es el más sencillo.

Como cambiar el tamaño de las ventanas.

Si el apuntador (o puntero) está dentro de una ventana y es desplazado a una de las orillas, adoptará la forma de flecha. Esta señalará la dirección en la que usted podrá cambiar el tamaño de la ventana. Si se apuntara en una de las esquinas, podría cambiar el tamaño de los dos lados que confluyan en la esquina. Para cambiar el tamaño con el apuntador en forma de flecha, mantenga pulsado el botón izquierdo del ratón, arrastre el borde de la ventana hasta que ésta adquiera el tamaño deseado y suelte el botón. Si el tamaño no fuese el que desea repita la operación.

El menú de ventana.

Con ayuda de **mwm**, cada ventana se puede controlar mediante su propio *menú de ventana*. Hay muchas formas de tener un menú de ventana. Dos de ellas son las siguientes: hacer clic en el botón del menú que está en el extremo superior

izquierdo de un marco (figura 4.4.) y hacer clic sobre el ícono. En la figura 4.5. se muestra un menú de ventana.

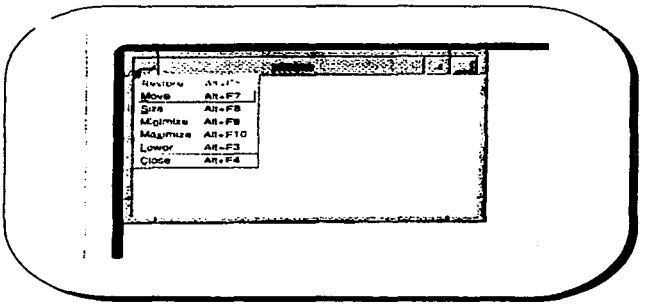

figura 4.5. El menú de ventana mwm.

Cuando aparezca el menú, podrá señalar uno de sus elementos y hacer clic en él.

- **Restore (Restaurar)** Devuelve al tamaño original una ventana que haya reducido al mínimo (a ícono) o ampliado al máximo. En la figura 4.5, la entrada **Restore** aparece en un tono gris, lo que significa que no se puede seleccionar. (Debido a que la ventana está en su dimensión normal, no se puede restaurar porque su tamaño no se ha alterado).
- **Move (Mover)** Vuelve a poner una ventana en la pantalla.

- **Resize (Tamaño)** Modifica el tamaño de una ventana.
- **Minimize (Minimizar) y Maximize (Maximizar)** Estas operaciones se explicaron en "Cómo usar la barra de títulos".
- **Lower (Bajar)** Baja una ventana hasta la parte inferior de la pila, en el caso de que se encuentre en una pila de ventanas.
- **Close (Cerrar)** Cierra la ventana y termina el programa que ésta contenga. Úselo como último recurso. Si el programa cuenta con un menú especial o un comando *de salida*- por ejemplo, introduciendo **exit** en el indicador del shell de una ventana **xterm** , úselo en vez de **Close (Cerrar)**. (Vea la sección "Como salir", más adelante).

En el menú, luego de cada comando, aparece un *método abreviado de teclado.* No tiene que recurrir al ratón para escoger un comando. Por ejemplo, para **Minimizar** una ventana (reducirla al icono), sólo tiene que pulsar la tecla **ALT** (o **META)** y oprimir la tecla **F9** . El método abreviado para que aparezca el menú de ventanas (y también para quitarlo) consiste en mantener oprimida la tecla **SHIFT** (MAYUS) y pulsar la tecla **ESC** . Si en su teclado faltara alguna de estas teclas, es posible adaptar el menú para utilizar otras.

IV.3.3. La ventana XTERM

Uno de los tipos más importantes de ventana es el de la ventana **xterm** este permite contar con una ventana emuladora determinable en cuyo interior se realice una sesión de login de UNIX, exactamente como si se tratará de una

terminal en miniatura. Puede haber varias ventanas **xterm** abiertas al mismo tiempo y, en cada una de ellas realizarse algo distinto para introducir un comando de UNIX o responder a un indicador cuando se está en una ventana, fije el foco en tal ventana y teclee. Los programas que estén en las otras ventanas seguirán ejecutándose; si estos necesitarán alguna intervención por parte de usted, esperarian como si se estuvieran realizando en una terminal separada.

En la figura 4.2D y en la figura 4.4 se observa una ventana **xterm** sencilla, en cuyo interior aparece el indicador del shell (%). Si usted introduce un comando de UNIX (por ejemplo, date) se ejecutará como lo haría si estuviera en una terminal sin sistema de ventanas.

También es posible iniciar programas de ventanas independientes y basados en X (comúnmente conocidos como clientes) con sólo introducir comandos en una ventana **xterm.** Si bien usted puede iniciar nuevos clientes (**xterm , xcalc,** etc) desde cualquier ventana **xterm** que este abierta en su computadora, es recomendable que los inicie desde la primera ventana que haya abierto. Lo anterior y el controlador de tareas de shell le permitirán localizar y controlar con gran facilidad todos los clientes.

Un ejemplo de esto. Para iniciar la calculadora conocida como **xcalc** introduzca:

% xcalc &

(1) 12345

El shell imprimirá un número **PID**, como 12345. si omitió el signo (&) al final de la línea, salga de **xcalc** utilizando el carácter de interrupción correspondiente (por ejemplo **CTRL-C**) para que así aparezca otro indicador del shell luego, proceda a introducir correctamente el comando.

El posible colocar la nueva ventana y fijar el foco automáticamente puede suceder también que la ventana (o un esquema se está) aparezca " flotando " en pantalla siguiendo al apuntador, hasta que usted apunte a un sitio determinado y haga clic con el botón del ratón para fijar definitivamente la ventana.

Es posible también iniciar una nueva **xterm** desde una **xterm** ya existente. Introduzca **xterm &** (no se olvide de poner este signo) en el indicador del shell. Este mismo método sirve también para iniciar otros programas de X.

IV.3.4. El menú raíz o principal.

Si usted traslada el apuntador a la ventana principal (el " escritorio " o " desktop " que está detrás de todas las ventanas) y oprime el botón respectivo del ratón (el izquierdo o el derecho, dependiendo de como lo haya configurado), deberá aparecer el menú raíz (root menú). Hay veces que es necesario mantener presionado el botón mientras se quiera ver el menú. El menú raíz cuenta con

comandos para controlar las ventanas. Los comandos del menú aveces cambian de un sistema a otro.

El administrador del sistema (o usted mismo, en caso de que estudie su administrador de ventanas) puede añadir comandos al menú raiz, como operaciones del administrador de ventanas o comandos para abrir otras ventanas. Por ejemplo, un elemento del menú " **New Window** " (ventana nueva) sirve para abrir una nueva ventana **xterm** . El elemento " calculator " (calculadora) le servirá para iniciar **xcalc.**

IV.4. OTROS CLIENTES X.

A continuación listamos algunos de los programas de cliente X con los que posiblemente cuente su sistema:

- **resize:** ayuda a los programas de ventanas **xterm** a reconocer un nuevo tamaño de ventana
- **xbiff:** le indica si tiene correo electrónico
- **xclipboard:** ayuda a copiar y pegar texto
- **xdpr:** para imprimir una ventana (mediante la impresora)
- **xedit:** editor de textos sencillo
- **xmag:** amplia algunas partes de la pantalla
- **xman:** examina páginas del manual (de consulta)
- **xmh:** programa de correo electrónico

- **xset:** configura las preferencias del usuario

Para mayor información acerca de estos programas, consulte la documentación en línea (en pantalla) o en la guía del usuario de X Window.

Cómo salir.

Como casi todo en X, es posible configurar la forma de salir del programa. La clave para hacerlo reside en saber cuál de los programas (en las ventanas o en el administrador de ventanas) es el programa controlador. Cuando se sale de este último, de inmediato se cierran los restantes programas de X. El programa controlador suele ser el administrador de ventanas o la ventana **xterm** con la que se inicia la sesión X.

Programa para salir.

Si su programa controlador es una ventana **xterm** , es recomendable que deje esa ventana del tamaño de un icono desde que inicia la sesión hasta que haya cerrado todos los otros clientes X. Estos evitará que finalice la sesión en X si, por accidente, cierra la ventana **xterm** anticipadamente.

Para salir del administrador de ventana, escoja el comando **Exit o Quit (Salir)** del menú principal.

Estos son los pasos para cerrar X:

1. Deje todos los programas que no son de control (los programas que no sean el programa de control).

   Si en algunas ventanas se ejecutan programas que cuentan con sus propios comandos para "salir", sería recomendable usarlos. Por ejemplo, si usted trabaja en un editor de texto de una ventana **xterm** , use el comando "quit" (salir), utilice **Close** del menú de ventana.

   Si en algunos iconos se ejecutan programas que tienen sus propios comandos para "salir", abra los iconos como ventanas y use los comandos "quit" (salir) respectivos.

2. Salir del programa de control.

   Luego de cerrar X, posiblemente aparezca el indicador del shell de UNIX. De ser así, para finalizar la sesión introduzca **exit** (salir). Si aparece otra caja de inicio de sesión desde **xdm** (como en la figura 4.2A) quiere decir que ya logró terminar la sesión.

El sistema operativo Unix SunOS es una combinación del Unix de Berkey y del Sistema V de AT&T. De hecho, AT&T en Conjunción con Sun Microsystems (la compañía que fabrica las computadoras Sun) están desarrollando las futuras versiones de Unix.

SunOS provee dos herramientas que permiten ejecutar comandos en una forma

más amena para el usuario: Alias y History.

IV.5. FACILIDADES DE ALIAS.

SINTAXIS: alias ({new-command} {unix-command})

Debido a que los comandos de Unix son muy difíciles de memorizar (por sus extraños nombres) , SunOS permite "mapear" los comandos de Unix hacia otros más fáciles. Por ejemplo, en Unix para hacer un "dir" de "DOS" se usa el comando "ls" (que significa "listar"). Otro ejemplo es: para ver el contenido de un archivo (comando "type" en una Pc con DOS) en Unix el comando es "cat" o el comando "more".

Si queremos usar "type" para ver un archivo, simplemente usamos la facilidad de "alias" de sunOS tecleando:

alias type more <ret>
Y ya cada vez que tecleemos:
type { file } < ret >

SunOS sabrá que tiene que ser un "more" cuando el usuario teclee "type". Si se teclea la palabra alias sin argumento, el sistema desplegará los alias que ya existen.

Podemos tener más alias útiles tales como:

alias dir `ls -la'
alias type `more'
alias lo"exit'
alias print `lpr'
alias del `rm'
alias copy `cp'
alias rename `mv'
alias ren `mv'
etcétera, etcétera.

Si el alias lo tecleamos en el prompt, cuando el usuario abandone su cuenta tal alias desaparece y sería necesario volver a teclearlo para usarlo otra vez.

Para hacer que un alias sea permanente existe un archivo que se ejecuta automática cada vez que el usuario entra a la computadora, ese archivo se llama ".cshrc" y puede editarse por el usuario.

Generalmente el Administrador del Sistema usa ese archivo para ponerle al usuario su "PATH" y otras variables de ambiente.

Si quieres editar tus propios " alias", pon los en ese archivo usando el editor que más te guste (por ejemplo "vi" o el "extedit" de "Sunview").

También es importante que sepas que en el archivo "/usuarios/Cshrc" esta el ambiente actualizado para usar toda la paquetería existente en Sun, por lo cual,

si tienes problemas de acceso a un paquete, recomiendo copiar ese archivo a tu directorio y le pongas como nombre ".cshrc".

IV.6. FACILIDADES DE HISTORY.

Otra facilidad de SunOS es la de "history" y nos permite volver a ejecutar un comando que ya se había tecleado anteriormente.

Supongamos la siguiente sesión de trabajo:

login: cursos
password: <----->
SunOS Release 4.1_ PSR_ A (GENERIC) #1: Fri Mar 8
10:20:52 CST 1991
cursos> ls <ret>
hola prueba.pas analisis
cursos> cat hola <ret>

MEMORANDUM

DE: Ing. Fill The Blank Space
PARA: Ing. The Blank Space
ASUNTO: El que se indica

cursos> rm hola <ret>

```
cursos> ls <re>
prueba.pas  análisis
cursos> history

1   ls
2   cat hola
3   rm hola
4   ls
5   history
cursos>
```

Como se ve, history sin argumentos despliega la lista de comandos que se han realizado hasta el momento. Lo útil es que se puede repetir un comando previo:

1) Tal como se tecleo anteriormente.
2) Con modificaciones

Por ejemplo, el comando "rm hola" en la lista de history es el número "3", si quiero volver a ejecutarlo tal cual esta, teclearía:

```
cursos> !3 <ret>
rm hola
cursos >_
```

Como se observa, para ejecutar un comando previo de history se teclea al carácter "!" pegado al número de comando en la lista de history. El sistema

imprime en pantalla que comando corresponde y lo ejecuta.

Si se quiere ejecutar un comando anterior pero con modificaciones, se teclea:
cursos > !3:s:hola:suma.pas <ret>
rm suma.pas
cursos >_

Es decir, "!3" quiere decir que voy a ejecutar el comando "3", ":s" significa que voy a sustituirle ":hola" la palabra "hola" por la palabra "suma.pas" (":suma.pas".

IV.7. UNIX: COMUNICACIONES REMOTAS.

En las máquinas unix existen varios comandos para lograr conexiones a otras computadoras: "rlogin", "telnet", "ftp", "rsh".

También existen algunos comandos para adquirir información acerca de una computadora o un usuario: "finger", "who", "w", "rusers".

Debido a que las redes unix tienen acceso a la red internacional, cada uno de sus elementos debe tener un nombre único a nivel mundial. Los nombres únicos se forman concatenando lo que se llama "nombre del dominio" al nombre local de la computadora. En la UDLA, el nombre de dominio es:
pue.udlap.mx

De manera que el nombre internacional de la Sun de la UDLA quedan del tipo: **cca.pue.udlap.mx, ccb.pue.udlap.mx, lara2.pue.udlap.mx,**

Comando "rlogin"
SINTAXIS: rlogin nombre-de-computadora [-l nombre-de-usuario

Significa "remote login" y sirve para conectarse a una computadora ya sea la misma en la que esta trabajando u otra en la red.

Rlogin sirve para conectarse a una computadora, ya sea con el mismo nombre de usuario que tenemos en la nuestra o como otro usuario.

Por ejemplo, si se está trabajando como usuario "cursos" y deseo entrar a "cca" como usuario "lcl" la sesión sería así:

cursos> rlogin cca -l lcl <ret>
password: <password-de-lcl>
SunOS Release 4.1_PSR_A (GENERIC) #1: Fri Mar 8 10:20:52: CST 1991 lcl >

Si se especifica el argumento "-l nombre-de-usuario", entonces "rlogin" nos conecta a la otra computadora como "cursos" (en grla., con el mismo nombre de usuario actual).

Recuerde que si la computadora no esta es la local, se debe usar el nombre a

nivel mundial (Ejemplo: cca.pue.udlap.mx).

Comando "rsh"
SINTAXIS: rsh nombre-de-computadora [-l nombre-de-usuario

comando Rsh (remote shell) sirve para ejecutar un comando en una computadora igual o diferente a la nuestra.

Por ejemplo, si estamos en "cch" y queremos ejecutar el comando "ls" en la computadora "ccj" teclearíamos:
cursos > rsh ccj ls <ret>

La salida del comando "ls" sale en la pantalla local, en "cch".

Si se quiere ejecutar un comando pero como si fuera otro usuario se usa la opción "-l nombre-de-usuario".

Si no especificamos el "comando ", el rsh funciona igual que "rlogin".

Comando "telnet"
SINTAXIS: telnet [{nombre-de-computadora} { IP Adress}

Telnet sirve para conectar a otra computadora que posiblemente no tiene Sistema Operativo Unix (Por ejemplo a la Vax).

Telnet debe ser el nombre de la computadora o su IP Address, Por ejemplo, para conectarnos a la "cca" puede ser así:

curos> telnet cca <ret>
O podría ser así:
cursos>telnet 140.148.1.18 <ret>

Recordemos que si es por nombre y la conexión es internacional, usaremos el nombre único mundial.

Comando "ftp"
SINTAXIS: ftp [{nombre-de-computadora} {IP Address}

El comando ftp 8File Transfer Protocol) sirve para que podamos enviar o traer archivos de una computadora a otra.

Los pasos con ftp son:

a) Conectarse a una computadora con ftp.
b) Ejecutar comandos para traer o enviar los archivos.
c) Cerrar la conexión.

En el mundo existen muchas computadoras con un usuario llamado "anonymous" que ofrece archivos para que cualquier otro usuario los adquiera, con la ventaja que ese usuario no necesita "password". Sin embargo, se

recomienda que al usar ese nombre de usuario se ponga como password la dirección de correo electrónico tuyo.

Los comandos que se pueden ejecutar dentro de ftp más importantes son:

1.- get (para traer un archivo)
2.- put (para escribir un archivo)
3.- mget (para traer múltiples archivos)
4.- mput (para escribir múltiples archivos)
5.- pwd (para saber en que directorio estamos)
6.- dir (para ver que archivos existen en el directorio actual)
7.- cd (para cambiar de directorio en la computadora remota)
8.- lcd (para cambiar de directorio en la computadora local)
9.- quit (para salir de ftp y cerrar la conexión).

Reanudando el inciso (a) podríamos dar algunos comandos

ftp> pwd <ret>
/pub/sun
ftp > dir <ret>
suma.pas eventos.txt tarea.psico
ftp > lcd/ usuarios/cursos/archivos-de-ftp <ret>
ftp > mget * <ret>
get suma.pas ? y <ret>
get eventos.txt ? n <ret>

get area.psico ? y <ret>
ftp > quit <ret>
cursos > _
Para cerrar la conexión, teclear "quit" en el prompt de "ftp".

Comando "w" y "who"
SINTAXIS: w
SINTAXIS: who

El comando "w" despliega en la pantalla cuales usuarios están trabajando en la computadora local y que actividad están realizando. Ejemplo:

cursos > w <ret>
11:53 am up 39 mins, 1 user, load average: 0.14,0.03,0.01

User	tty	login	idle	JCPU	PCPU	what
enrique	console	11:31am	21			textedit -Wp 479
enrique	ttyp0	11:32am	12	2	2	w
enrique	ttyp1	11:32am	21			-bin/csh
enrique	ttyp2	11:32am		26	17	vi cursocrem.xt

cursos >_

La columna "user" indica el usuario, "tty" indica en que dispositivo se esta trabajando, "login" indica a que hora entro el usuario a la computadora, "idle2 indica cuanto tiempo lleva sin hacer algo, etc.

El comando "who" es similar al "w", reporta que usuarios están trabajando localmente, en que dispositivo y a que horas entraron.

Comando "rusers"

El comando rusers reporta que usuarios están trabajando en la red y en que máquinas están haciéndolo.

Comando "finger"

Reporta información sobre un usuario o sobre una computadora. Para obtener información de un usuario teclear:

cursos > finger nombre-de-usuario <ret>
Para obtenerla de una computadora teclear:
cursos > finger nombre-de-computadora <ret>

Note el simbolo de " " que precede al nombre de computadora.

CAPITULO V. EDITOR DE PANTALLA.

Objetivo:

MOSTRAR UN RESUMEN DEL CÓDIGO FUENTE, QUE PROPORCIONE LAS CARACTERÍSTICAS PRINCIPALES DEL EDITOR DE PANTALLA PARA AMBIENTE UNIX.

CAPÍTULO V. EDITOR DE PANTALLAS (CÓDIGO FUENTE)

CURSES.C

```
# include "config.h"
# include "vi.h"
# if ANY_UNIX
# if UNIXV !! COH_386
# ifdef TERMIOS
# include <termios.h>
# else
# include <termio.h>
# endif
# ifndef NO_S5WINSIZE
# ifndef_SEQUENT_
# include <sys/stream.h>
# include <sys/ttycom.h>
# endif
# else
# undef  TIOCGWINSZ
# endif
# else
# include <sgtty.h>
# endif
# endif

# if TOS
# include <osbind.h>
# endif

# if OSK
 # include <sgstat.h>
# endif

# if VMS
extern int VMS_read_raw;
# endif

static void starttcap();
char *termtype;
short ospeed;
# if OSK
char PC_;
char *BC;
# else
char PC;
# endif
WINDOW *stdscr;
WINDOW  kbuf[KBSTZ];
int LINES;
int COLS;
int AM;
int PT;
char *VB;
char *UP;
char *SO="";
char *SE="";
char *US="";
char *UE="";
char *MD="";
char *ME="";
char *AS="";
char *AE="";
# ifndef NO_VISIBLE
char *MV;
```

```
 #endif
 char *CM;
 char *CE;
 char *CD;
 char *AL;
 char *DL;
 # if OSK
 char *SR_;
 # else
 char *SR;
 #endif
 char *KS="";
 char *KE="";
 char *KU;
 char *KD;
 char *KL;
 char *KR;
 char *HM;
 char *EN;
 char *PU;
 char *PD;
 char *KI;
 #ifndef NO_FKEY
 char *FKEY[NFKEYS];
 #endif
 char *IM="";
 char *IC="";
 char *EI="";
 char *DC;
 char *TI="";
 char *TE="";
 #ifndef NO_CURSORSHAPE
 #if 1
 char *CQ=(char *)0;
 char *CX=(char *)1;
 char *CV=(char *)2;
 char *CI=(char *)3;
 char *CR=(char *)4;
 #else
 char *CQ="";
 char *CX="";
 char *CV="";
 char *CI="";
 char *CR="";
 #endif
 #endif
 char *aend="";
 char ERASEKEY;
 #ifndef NO_COLOR
 char normalcolor[24];
 char SOcolor[24];
 char SEcolor[24];
 char UScolor[24];
 char UEcolor[24];
 char MDcolor[24];
 char MEcolor[24];
 char AScolor[24];
 char AEcolor[24];
 # ifndef NO_POPUP
 char POPUPcolor[24];
 #endif
 #ifndef NO_VISIBLE
 char VISIBLEcolor[24];
 #endif
 #endif

 # if ANY_UNIX
 # if UNIXV|| COH_386
 # ifdef TERMIOS
 static struct termios      oldtermio;
 static struct termios      newtermio;
```

```
# else
static struct termio        oldtermio,
static struct termio        newtermio,
# endif
# else
static struct sgttyb        oldsgttyb,
static struct sgttyb        mewsgttyb;
static int oldint,
# ifdef TIOCSLTC
static int          oldswitch,
static int          olddswich;
static int          oldquote,
# endif
# endif
# endif

# if OSK
static struct sgbuf         oldsgttyb,
static struct sgbuf         newsgttyb;
# endif
static char *capbuf,

void initscr()
{
        temtype=getenv("TERM");
#if VMS
        if (!strcmp(termtype,"unknown"))
        termtype=getenv("STING_TERM"),
# endif
# if MSDOS
        if (!termtype)
        ;
# ifdef RAINBOW
if (*(unsigned char far*)(0x0000eL))==6
|| *(unsigned char far*)(0x0000eL))== 148)
        {
                termtype="rainbow";
        }
        else
# endif
        termtype="pcbios";
        }
if (!strcmp(termtype,"pcbios"))
# else
        if (!termtype)
# endif
        {
# if ANY_UNIX
        write(2,"Debe definirse la variable de
ambiente TERM\n", (unisgned)38)
        exit(2),
# endif
#if OSK
        write(2,"Debe definirse la variable de
ambiente TERM\n", (unisgned)38)
        exit(2);
# endif
# if AMIGA
        termtype=TERMTYPE;
        starttcap(termtype);
# endif
# if MSDOS
        starttcap("pcbios"),
# endif
# if TOS
        termtype="vt52";
        starttcap(termtype),
# endif
# if VMS
        write(2,"Terminal no definida
STING_TERM\n",(unsigned)36),
        exit(2),
# endif
```

```
        }
        else
        {
# if MSDOS
                *o_pcbios=0;
# endif
                startlcap(termtype);
        }
        stdscr=kbuf;

# if ANY_UNIX
# if UNIXV||COH_386
# ifdef TERMIOS
        togetattr(2,&oldtermio);
# else
        ioctl(2,TCGETA,&oldtermio);
# endif
# endif
# if OSK
        _gs_opt(0,&oldsgttyb);
# endif
# if VMS
        VMS_read_raw=1;
        vms_open_tty();
# endif
        resume_curses(TRUE);
}
void endwin()
{
        suspended_curses();
# if AMIGA
        amiclosewin();
# endif
}
static int curses_active=FALSE;
extern int oldeurs;

void suspend_curses()
{
# if ANY_UNIX && !(UNIXV||COH_386)
        struct tchars tbuf;
# ifdef TIOCSLTC
        struct ltchars lbuf;
# endif
# endif
# ifndef NO_CURSORSHAPE
        if (has_CQ)
        {
                do_CQ();
                oldcurs=0;
        }
# endif
        if (has_TE)
        {
                do_KE();
        }
# ifndef NO_COLOR
        quitecolor();
# endif
        refresh();

# if ANY_UNIX
# if (UNIXV||COH_386)
# ifdef TERMIOS
        tcsetattr(2,TCSADRAIN,&oldtermio);
# else
        ioctl(2, TCSETAW,&OLDTERMIO);
# endif
# else
        ioctl(2,TIOCSETP,&oldsgttyb);
ioctl(2,TIOCGETC,(struct sgttyb *)&tbuf);
tbuf.t_intre=oldint;
```

```
ioctl(2, TIOCSETC,(struct sgttyb *)&tbuf);
# ifdef TIOCSLTC
        ioctl(2,TIOCGLTC,&ltbuf);
        ltbuf.t_suspe=oldswitch;
        ltbuf.t_dsuspe=olddswitch;
        ltbuf.t_lnexte=oldquote;
        ioctl(2,TIOCSLTC,&ltbuf);
# endif
# endif
# endif
# if OSK
        _es_opt(0,&oldsgttyb);
# endif
# if AMIGA
        ttyshutdown();
# endif
# if MSDOS
        raw_set_stdio(FALSE);
# endif
# if VMS
        VMS_read_raw=0;
# endif
        curses_active=FALSE;
}
void resume_curses(quietly)
        int quietly;
{
        if (!curses_active)
        {
# if ANY_UNIX
# if UNIX||COH_386
        ospeed=(oldtermio.c_cflag & CBAUD);
        ERASEKEY=oldtermio.c_cc[VERASE];
        newtermio=oldtermio;
        newtermio.c_iflag &=
(IXON|IXOFF|IXANY|ISTRIP|IGNBRK);
        newtermio.c_oflag &=~OPOST;
        newtermio.c_lflag &=ISIG;
        newtermio.c_cc[VINTR]=ctrl('C');
        newtermio.c_cc[VMIN]=1;
        newtermio.c_cc[VTIME]=0;
# ifdef VSWTCH
        newtermio.c_cc[VSWTCH]=0;
# endif
# ifdef VSUSP
        newtermio.c_cc[VSUSP]=0;
# endif
# ifdef TERMIOS
        tcsetattr(2,TCSADRAIN,&newtermio);
# else
        ioctl(2,TCSETAW,&newtermio);
# endif
# else
        struct tchars tbuf;
# ifdef TIOCSLTC
        struct ltchars ltbuf;
# endif
        ospeed=oldsgttyb.sg_ospeed;
        ERASEKEY=oldsgttyb.sg_erase;
        newsgttyb=oldsgttyb;
        newsgttyb.sg_flags|=CBREAK;
        newsgttyb.sg_flags &=
~(CRMOD|ECHO|XTABS);
ioctl(2,TIOCGETC,(struct sgttyb *) &tbuf);
oldin=tbuf.t_intrc;
tbuf.t_intrc=ctrl('C');
ioctl(2,TIOCSETC,(struct sgttyb *)&tbuf);
# ifdef TIOCSLTC
ioctl(2,TIOCGLTC,&ltbuf);
oldswich=ltbuf.t_suspc;
ltbuf.t_suspc=0;
```

```
olddswitch=ltbuf.t_dsuspe;
ltbuf.t_dsuspe=0;
oldquote=ltbuf.t_lnextc;
ltbuf.t_lnextc=0;
ioctl(2,TIOCSLTC,&ltbuf);

newsgttyb=oldsgttyb;
newsgttyb.sg_echo=0;
newsgttyb.sg_eofch=0;
newsgttyb.sg_kbach=0;
newsgttyb.sg_kbich=ctrl('C');
_ss_opt(0,&newsgttyb);
ospeed=oldsgttyb.sg_baud;
ERASEKEY=oldsgttyb.sg_bspch;

ttysetup();
raw_set_stdio(TRUE);

VMS_read_raw=1;
        { int c;
                read(0,&c,0);
        }
        ERASEKEY='\177';
# endif
        curses_active=TRUE;
        }
        if (quietly)
        {
                if(has_TI)
        {
                do_TI();
        }
        if (has_KS)
        {
                do_KS();
        }
                return;
        }
        signal(SIGINT,SIG_IGN);
        move(LINES-1,0);
        do_SO();
# if VMS
qaddstr("\n[Press <RETURN> to continue ]");
# else
qaddstr("[Press <RETURN> to continue ]");
# endif
        do_SE();
        refresh();
        ttyread(kbuf,20,0);
        if (has_TI)
        {
                do_TI();
        }
        if (kbuf[0]='Y')
{
        mode=MODE_COLON;
        addch('\n');
        refresh();
}
else
{
        mode=MODE_VI;
        redraw(MARK_UNSET,FALSE);
}
        exwrote=FALSE;
        signal(SIGINT, trapint);
}
static void mayhave(T,s)
        char    **T;
        char    *s;
{
```

```
        char    *val;
        val=tgetstr(s,&capbuf);
        if(val)
        {
                *T=val;
        }
}
static void musthave(T,s)
        char    **T;
        char    *s;
{
        mayhave(T,s);
        if(!*T)
        {
write(2,"Esta entrada del termcap no tiene ",
(unsigned)30);
                write(2,s,(unsigned)2);
        write(2,"=:capacidad\n",(unsigned)14);
# if OSK
                write(2,"\l",1);
# endif
                exit(2);
        }
}
static void pair(T,U,sT,sU)
        char    **T;
        char    **U;
        char    *sT;
        char    *sU;
{
        mayhave(T,sT);
        mayhave(U,sU);
        if (!**T||!**U)
        {
                *T=*U="";
        }
}
static void starttcap(term)
        char    *term
{
        static char obmem[800];
        capbuf=obmem;
        switch(tgetent(kbuf,term))
        {
        case -1:
        write(2,"No puedo leer /etc/termcap\n",
(unsigned)24);
# if OSK
                write(2,"\l",1);
# endif
                exit(2);
        case 0;
write(2,"variable TERM no reconocida\n",
(unsigned)23);
# if OSK
                write(2,"\l",1);
# endif
                exit(3);
        }
        musthave(&UP,"up");
        mayhave(&VB,"vb");
        musthave(&CM,"cm");
        pair(&SO,&SE,"so","se");
        mayhave(&TI,"ti");
        mayhave(&TE,"te");
        if (tgetnum("ug")<=0)
        {
                pair(&US,&UE,"us","ue");
                pair(&MD,&ME,"md","me");
                pair(&AS,&AE,"as","ae");
                if(!*AS)
                {
```

```
                AS=US;
                AE=UE;
            }
        }
# ifndef NO_VISIBLE
        MV=SO;
        mayhave(&MV,"mv");
# endif
        mayhave(&AL,"al");
        mayhave(&DL,"dl");
        mayhave(&CE,"ce");
        mayhave(&CD,"cd");
# if OSK
        mayhave(&SR,"sr");
# else
        mayhave(&SR,"sr");
# endif
        pair(&IM,&EI,"im","ei");
        mayhave(&IC,"ic");
        mayhave(&DC,"dc");
        AM=(tgetflag("am")&& !tgetflag("xn"));
        PT=tgetflag("pt");
# if AMIGA
        amiopenwin(termtype);
        ttysetup();
# endif
        getsize(0);
        pair(&KS,&KE,"ks","ke");
        mayhave(&KU,"ku");
        mayhave(&KD,"kd");
        mayhave(&KR,"kr");
        mayhave(&KL,"kl");
        if (KL&&KL[0]=='\b'&& !KL[1])
        {
            KL=(char *)0;
        }
        mayhave(&PU,"kP");
        mayhave(&PD,"kN");
        mayhave(&HM,"kh");
        mayhave(&EN,"kH");
        mayhave(&KI,"kI");
# ifndef CRUNCH
        IF (!PU) mayhave(&PU,"k2");
        IF (!PD) mayhave(&PD,"k5");
        IF (!HM) mayhave(&HM,"k1");
        IF (!EN) mayhave(&EN,"k4");
        mayhave(&PU,"PU");
        mayhave(&PD,"PD");
        mayhave(&HM,"HM");
        mayhave(&EN,"EN");
# endif
# ifndef NO_FKEY
        mayhave(&FKEY[0],"k0");
        mayhave(&FKEY[1],"k1");
        mayhave(&FKEY[2],"k2");
        mayhave(&FKEY[3],"k3");
        mayhave(&FKEY[4],"k4");
        mayhave(&FKEY[5],"k5");
        mayhave(&FKEY[6],"k6");
        mayhave(&FKEY[7],"k7");
        mayhave(&FKEY[8],"k8");
        mayhave(&FKEY[9],"k9");
# if ifndef NO_SHIFT_FKEY
        mayhave(&FKEY[10],"s0");
        mayhave(&FKEY[11],"s1");
        mayhave(&FKEY[12],"s2");
        mayhave(&FKEY[13],"s3");
        mayhave(&FKEY[14],"s4");
        mayhave(&FKEY[15],"s5");
        mayhave(&FKEY[16],"s6");
        mayhave(&FKEY[17],"s7");
```

```
        mayhave(&FKEY[18],"s8");
        mayhave(&FKEY[19],"s9");
# indef NO_CTRL_FKEY
        mayhave(&FKEY[20],"c0");
        mayhave(&FKEY[21],"c1");
        mayhave(&FKEY[22],"c2");
        mayhave(&FKEY[23],"c3");
        mayhave(&FKEY[24],"c4");
        mayhave(&FKEY[25],"c5");
        mayhave(&FKEY[26],"c6");
        mayhave(&FKEY[27],"c7");
        mayhave(&FKEY[28],"c8");
        mayhave(&FKEY[29],"c9");
# ifndef NO_ALT_FKEY
        mayhave(&FKEY[30],"a0");
        mayhave(&FKEY[31],"a1");
        mayhave(&FKEY[32],"a2");
        mayhave(&FKEY[33],"a3");
        mayhave(&FKEY[34],"a4");
        mayhave(&FKEY[35],"a5");
        mayhave(&FKEY[36],"a6");
        mayhave(&FKEY[37],"a7");
        mayhave(&FKEY[38],"a8");
        mayhave(&FKEY[39],"a9");
# endif
# endif
# endif
# endif
# ifndef NO_CURSORSHAPE
        CQ=tgetstr("cQ",&capbuf);
        if (has_CQ)
        {
        CX=tgetstr("cX",&capbuf);
        if (!CX) CX=CQ;
        CV=tgetstr("cV",&capbuf);
        if (!CV) CV=CQ;
        CI=tgetstr("cI",&capbuf);
        if (!CI) CI=CQ;
        CR=tgetstr("cR",&capbuf);
        if (!CR) CR=CQ;
}
# ifndef CRUNCH
        else
        {
                CQ=CV="";
                pair(&CQ,&CV,"ve","vs");
                CX=CI=CR=CQ;
        }
# endif
# endif
# ifndef NO_COLOR
        strcpy(SOcolor,SO);
        strcpy(SEcolor,SE);
        strcpy(AScolor,AS);
        strcpy(AEcolor,AE);
        strcpy(MDcolor,MD);
        strcpy(MEcolor,ME);
        strcpy(UScolor,US);
        strcpy(UEcolor,UE);
# ifndef NO_POPUP
        strcpy(POPUPcolor,SO);
# endif
# ifndef NO_VISIBLE
        strcpy(VISIBLEcolor,MV);
# endif
# endif
}
int getsize(signo)
        int     signo;
{
        int     lines;
```

```
        int     cols,
# ifdef TIOCGWINSZ
        struct winsize size,
# endif
# ifndef SIGWINCH
        signal(SIGWINCH,(void *)getsize),
# endif
        lines=cols=0,
# ifdef TIOCGWINSZ
        IF (ioctl(2,TIOCGWINSZ,&size)>=0)
        {
                lines=size.ws_row,
                cols=size.ws_col,
        }
# endif
# if AMIGA
        if(!strcmp(TERMTYPE,termtype))
        {
        auto long len;
        auto char buf[30],
        Write(Output(),"\2330 q",4);
        len=Read(input(),buf,29);
        buf[len]='\000';
        sscanf(&buf[5],"%d;%d",&lines,&cols);
        }
# endif
        if((lines==0|| cols==0)&& signo ==0)
        {
                LINES=tgetnum("li");
                COLS=TGETNUM("co"),
        }
# if MSDOS
# ifdef RAINBOW
        if (!strcmp(termtype,"rainbow"))
        {
cols=*(unsigned char far *)0xEE000F57L,
        }
        else
# endif
        {
                lines=v_rows();
                cols=v_cols();
        }
# endif
        if lines >= 2 && cols();
        }
# endif
        if (lines>=2 &&cols >=30)
        {
                LINES=lines;
                COLS=cols;
        }
        if (LINES< 2|| COLS < 30)
        {
                write(2,"Pantallademasiado
pequeña\n",(unsigned)17),
# if OSK
                write(2,"\l",1);
# endif
                endwin();
                exit(2);
        }
# if AMIGA
if(*o_lines |= LINES|| *o_columns |= COLS)
        {
                *o_lines=LINES,
                *o_columns=COLS,
        }
# endif
        return 0,
}
```

```
int faddch(ch)
        int     ch;
{
        addch(ch);
        return 0;
}
void qaddstr(str)
        char    *str;
{
        REG char *s_, *d_;
# if MSDOS
        if (o_pcbios[0])
        {
                while(*str)
                        qaddch(*str++);
                return;
        }
# endif
        for (s_=(str),d_=stdscr,*d_++=*s_++)
        {
        }
        stdscr=d_-1;
}
void attrset(a)
        int     a;
{
        do_send();
        if (a==A_BOLD)
        {
                do_MD();
                send=ME;
        }
        else if (a==A_UNDERLINE)
        {
                do_US();
                send=UE;
        }
        else if (a ==A_ALTCHARSET)
        {
                do_AS();
                send=AE;
        }
        else
        {
                aend="";
        }
}
void insch(ch)
        int     ch;
        if (has_IM)
                do_IM();
        do_IC();
        qaddch(ch);
        if (has_EI)
                do_EI();
}
void wrefresh()
{
        if(stdscr-kbuf>2000)
        {
                VOIDBIOS(stdscr=kbuf,
                {
        ttywrite(kbuf,(unsigned)(stdscrkbuf));
                stdscr=kbuf;
                });
        }
}
# ifndef NO_COLOR
int ansiquit()
{
if(!strcmp(UP,"\033[A]")&&
strcmp(SOcolor,SO))
```

```
        {

        tputs("\033[37;40m\033[m|",1,faddch);
        clrtoeol();
        return 1;
        }
        return 0;
}
int ansicolor(cmode,attrbyte)
        int     cmode;
        int attribyte;
{
        char    temp[24];
if(strcmp(UP,"\033[A")&&
strcmp(UP,"\033OA"))
        {
                if (tmpfd>=0)
                        msg("no se como poner
los colores para esta terminal");
                return FALSE;
        }
# ifdef MWC
sprintf(temp,"\033[m\033[3%cm\033\4%cm%s%s
",04262537"  [  attrbyte&  0X07],"04261537"
[(attribyte >> 4 ) & 0x07], (attrbyte & 0x08)?
"\033[1m":"", (attrbyte & 0x80)? "\033[5m" " " );
# else
sprintf(temp,"\033[m\033[3%c;4%c%s%sm",04261537"  [ attrbyte& 0X07],"04261537"  [(attribyte
>> 4 ) & 0x07], (attrbyte & 0x08)?  "1":"",
(attrbyte & 0x80)? "5" " " ),

°endif
        switch(cmode)
        {
        case A_NORMAL;
        if(!strcmp(MEcolor,normalcolor))
                strcpy(MEcolor,temp);
        if(!strcmp(UEcolor,normalcolor))
                strcpy(UEcolor,temp);
        if(!strcmp(AEcolor,normalcolor))
                strcpy(AEcolor,temp);
        if(!strcmp(SEcolor,normalcolor))
                strcpy(SEcolor,temp);
        strcpy(normalcolor,temp);
        tputs(normalcolor,1,faddch);
        break;

case A_BOLD;
        strcpy(MDcolor,temp);
        strcpy(MEcolor,normalcolor);
        break;
case A_UNDERLINE;
        strcpy(UScolor,temp);
        strcpy(UEcolor,normalcolor);
        break;
case A_ALTCHARSET;
        strcpy(AScolor,temp);
        strcpy(AEcolor,normalcolor);
        break;
case A_STANDOUT;
        strcpy(SOcolor,temp);
        strcpy(SEcolor,normalcolor);
        break;
# indef NO_POPUP;
        strcpy(POPUPcolor,temp);
        break;
# endif
# ifndef NO_VISIBLE
        case A_VISIBLE;
                strcpy(VISIBLEcolor,temp);
        break;
#endif
        }
```

```
        return TRUE;
}
int
endcolor()
{
        if (aend==ME)
                tputs(MEcolor,1,faddch);
        else if (aend==UE)
                tputs(UEcolor,1,faddch);
        else if (aend==AE)
                tputs(AEcolor,1,faddch);
        else if (aend==SE)
                tputs(SEcolor,1,faddch);
        aend="";
        return0;
}
# endif
```

CONFIG.H

```
# ifndef_CONFIG_H
# define_CONFIG_H


# define M_SYSV

# ifdef  amiga
# define AMIGA              1
# define COMPILED_BY  "Manx Aztec C5.2b"
# define TINYSTACK 1
# endif



# ifdef   M_SYSV
# define UNIXV              1
# ifdef m_XENIX
# ifnder M_1386
# define TINYSTACK 1
# endif
# endif
# undef COHERENT
# endif



# ifdef   xelos
# define UNIXV              1
# endif



# ifdef bsd
# define BSD                1
# else
# ifdef sun
#  ifndef M_SYSV
# define BSD                1
# endif
# endif
# endif


# ifdef   M_186
# ifndef M_SYSV
# define MSDOS              1
# ifdef IBMC2
# define COMPILED_BY ."IBM C/2 1.00"
# else
# define MICROSOFT  1
#  define COMPILED_BY          "Microsoft   C
5.10"
# endif
#  define TINYSTACK  1
# endif
```

```
# endif


# ifdef   _TURBOC_
# define MSDOS                  1
# define TURBOC                 1
# ifdef_BORLANDC_               "Borland
C2.00"
# else
# define COMPILED_BY
(_TURBOC_>=661  ?"Turbo  C++  1.00":"Turbo
C2.00")
# endif
# define TINYSTACK    1
# endif



# ifdef   M68000
# define TOS 1
# define COMPILED_BY                    "Mark
Williams C"
# define TINYSTACK    1
# endif


# ifdef_atarist_
# ifdef_gem_ define TOS  1
# define COMPILED_BY            "GNU-
C"_VERSION_
# define TINYSTACK    1
# endif
# endif


# ifdef   OSK
# define COMPILED_BY            "Microware  C
V2.3 Edition 40"
# define TINYSTACK    1
# endif
```

```
# ifdef VMS
# define COMPILED_BY "VAX/VMS VAXC
compiler"
# undef VMS
# define VMS 1
# endif


# ifdef COHERENT
# ifdef_i386
# define COH_386  1
# define COH_286  0
# else
# define COH_386  0
# define COH_286  1
# endif
# undef COHERENT
# define COHERENT 1
# endif


# ifndef          BSD
# define BSD              0
# endif


# ifdef           UNIXV
# define UNIXV            0
# endif


# ifdef           UNIX7
# define UNIX7            0
# endif


# ifdef           MSDOS
# define MSDOS            0
# endif
```

```
# ifndef          TOS
# define TOS              0
# endif

# ifndef          AMIGA
# define AMIGA            0
# endif

# ifndef OSK
# define OSK              0
# endif

# ifndef COHERENT
# define COHERENT 0
# endif

# ifndef RAINBOW
# define RAINBOW 0
# endif

# ifndef VMS
# define VMS 0
# endif

# if IBSD  && IUNIXV && IUNIX7 && IMSDOS && ITOS && IAMIGA && IOSK && ICOHERENT && IVMS
# define MINIX            1

# else
# define MINIX 0
# endif

# if UNIXV|| UNIX7|| BSD|| MINIX|| COHERENT
# define ANY_UNIX 1
# else
# define ANY_UNIX 0
# endif

# ifndef TINYSTACK
# define TINYSTACK 0
# endif

# ifndef          AZTEC_C
# define aztec_c          0
# endif

# ifndef          MICROSOFT
#define MICROSOFT       0
# endif

# ifndef          TURBOC
# define TURBOC                   0
# endif

# ifdef_STDC_
# define NEWSTYLE 1
# endif
# ifdef_cplusplus
# define NEWSTYLE 1
# endif
# ifndef NEWSTYLE
# define NEWSTYLE 0
# endif

# if NEWSTYLE
# define P_(s) s
# else
# define P_(s) ()
# endif
# define VERSION "versión 1.0"
# if MSDOS
```

```
# define CREDIT "omar Marquina"
# if RAINBOW
#  define  CREDIT2  "Universidad  Autonoma
Metropolitana"
# endif
# endif

# if AMIGA
# define CREDIT "Omar Marquina"
# endif

# if TOS
# define CREDIT  "Omar Marquina"
# endif

# if OSK
# define CREDIT            "Omar Marquina"
# endif

# if CORERENT
# define CREDIT            "Omar Marquina"
# endif

# if VMS
# define CREDIT "Omar Marquina"
# endif

# if TOS && 1AMIGA
 define ttywrite(buf,len)write( 1,buf,(unsigned)(len))
# endif

# if BSD || unix7 || osk
# define strchr              index
# define strchr rindex
# endif
# if INEWSTYLE
# extern char *strchr(),
# endif

# if BSD
# ifndef_386BSD_
# define getowd              getwd
# endif
# endif
# if COH_286
# define getowd  getwd
# endif
extem char *getowd(),

# if ITOS
# define tread(fd,buf,n)
                read(fd,buf,(unsigned)(n))
#definetwrite (fd,buf,n)  write(fd,buf,(unsigned)(n))
# endif

# if UNIX7 ||TOS
# define void int
# endif

# if UNIX7 || MINIX
# define UCHAR(c)        ((c)& 0xff)
# define uchar                  char
# define UCHAR(c)        ((unsigned char)(c))
# define uchar                  unsigne char
# endif

# ifndet  -STDC_
# if BSD || AMIGA|| MINIX
extem void *malloc(),
# define _free_(ptr)        free((char *)ptr)
# endif
# else
```

```
# define_free_(ptr)        free((void *ptr)
# endif

# if !AMIGA
extern long lseek();
# endif

# ifndef_STDC_
extern char *getenv();
# endif

# ifdef_STDC_
# define SIGTYPE void
# else
# if MSDOS
# define SIGTYPE void
# else
# if UNIXV
# define SIGTYPE void
# endif
# endif
# ifndef SIGTYPE
# define SIGTYPE int
# endif

# if ANY_UNIX
# ifndef TMPDIR
# if MINIX
# define TMPDIR          "/usr/tmp"
# else
# define TMPDIR          "/var/tmp"
# endif
# endif
# ifndef PRSVDIR
# define PRSVDIR         "/var/preserve"
# endif
# ifndef PRSVINDEX
# define PRSVINDEX       "/var/preserve/Index"
# endif
# ifndef EXRC
# define EXRC            ".exrc"
# endif
# define SCRATCHOUT 110/os/soXXXXXX"
# ifndef SHELL
# define SHELL           "/bin/sh"
 endif
# if COHERENT
# ifndef REDIRECT
# define REDIRECT        ">"
# endif
# endif
 define gethome(x)       getenv("HOME")
# endif

# if AMIGA
# ifndef CC_COMMAND
# define CC_COMMAND
# endif
# ifndef COLON
# define COLON
# endif
# ifndef SYSEXRC
# define SYSEXRC         "S."EXRC
# endif
# ifndef MAXRCLEN
# define MAXRCLEN        2048
# endif
# ifndef NBUFS
# endif                                10
# endif
# ifndef NEEDSYNC
# define NEEDSYNC        TRUE
```

```
# endif
# ifndef PRSVDIR
# define PRSVDIR        "Elvis."
# endif
# ifndef PRSVINDEX
# define PRSVINDEX      "Elvis.Index"
# endif
# ifndef REDIRECT
# define REDIRECT       ">"
# endif
# ifndef SCRATCHIN
# define SCRATCHIN      110/oSSIXXXXXX"
# endif
# ifndef SCRATCHOUT
# define SCRATCHOUT 110/oSSOXXXXXX"
# endif
# ifndef SHELL
# define SHELL                  "newshell"
# endif
# ifndef TERMTYPE
# define TERMTYPE       "amiga"
# endif
# ifndef TMPDIR
# define TMPDIR         "T:"
# endif
# ifndef TMPNAME
# define TMPNAME        110/oSelv_%x.%x"
# endif
# define gethome(x)     getenv("HOME")
# endif

# if MSDOS || TOS

# ifndef TMPDIR
# define TMPDIR         "C:\\tmp"
# endif
# ifndefPRSVDIR
# define PRSVDIR        "C:\\preserve"
# endif
# ifndef PRSVINDEX
# define PRSVINDEX      "C:\\preserve\\Index"
# endif
# define TMPNAME        110/oS\\elv_%x.%x"
# if MSDOS
# if MICROSOFT
# define CC_COMMAND             "cl-o"
# else
# if_BORLANDC_
# define CC_COMMAND             "bcc"
# else
# if TURBOC
# define CC_COMMAND             "tcc"
#  endif
#  endif
#  endif
# endif
# define SCRATCHIN      110/oS\\sIXXXXXX"
# define SCRATCHOUT 110/oS\\soXXXXXX"
# define SLASH                  '\\'
# ifndef SHELL
# if TOS
# define SHELL          "shell ttp"
# else
# define SHELL          "command.com"
#  endif
# endif
# define NEEDSYNC       TRUE
# if TOS && _GNUC
# define REDIRECT       ">>"
# define CC_COMMAND             "gcc-c"
# else
# define REDIRECT       ">"
```

```
# endif
# endif
# if VMS
# ifndef TMPDIR
# define TMPDIR          "sys$scratch:"
# endif
# define TMPNAME         110/oSelv_%x %x;1"
# define SCRATCHIN       110/oSsiXXXXXX"
# define SCRATCHOUT      110/oSsoXXXXXX"
# define SLASH "."
# ifndef SHELL
# define SHELL "***"
# endif
# define REDIRECT        ">"
#                define                 tread(fd,buf,n)
vms_read(fd,buf,(unsigned)(n))
# define close vms_close
# define lseek vms_lseek
# define unlink vms_delete
# define deletedelete
# define rpipe vms_rpipe
# define rpelose vms_rpelose
# define ttyread vms_ttyread
# define gethome(x) getenv("HOME")
# define sync()
# define m_fWord m_fw_ord
# define m_bWord m_bw_ord
# define m_e Word m_ew_ord
# define m_Nsrch m_n_srch
#  define m_Fch  m_f_ch
# define m_Tch  m_t_ch
# define v_Ncharv_x_char
# define LINES elvis_LINES
# define COLS elvis_COLS
# define curser elvis_curser
# define stdser elvis_stdser
# define initser elvis_initser
# define endwin elvis_endwin
# define wrefresh elvis_wrefresh
# endif

# if OSK
#ifndef TMPDIR
# define TMPDIR          "/dd/tmp"
# endif
# ifndef PRSVINDEX
#define                          PRSVINDEX
"7dd/usr/preserve/Index"
# endif
# ifndef CC_COMMAND
# define CC_COMMAND          "cc-r"
# endif
# ifndef EXRC
# define EXRC                      ".exrc2
# endif
# define SCRATCHOUT 110/oSXXXXXX"
# ifndef SHELL
# define SHELL                     "shell2
# endif
# define FILEPERMS       (S_IREAD|S_IWRITE)
# define REDIRECT        ">>_"
# define syne()
# define gethome(x)       getenv("HOME")
# endif

# ifndef            TAGS
# define  TAGS                     "tags"
# endif

# ifndef TNPNAME
#  define TMPNAME       110/oSelv_%x %x"
# endif
```

```
# ifndef EXINIT
# define EXINIT                 "EXINIT"
# endif

# ifndef        EXRC
# define EXRC                   "elvis.rc"
# endif

# ifdef HMEXRC
# define HMEXRC                 EXRC
# endif

# ifndef        KEYWORDPRG
# define KEYWORDPRG             "ref"
# endif

# ifndef        SCRATCHOUT
# define SCRATCHIN      110/oS/SIXXXXXX"
# define SCRATCHOUT     110/oS/SOXXXXXX"
# endif

# ifndef ERRLIST
# define ERRLIST        "errs"
# endif

# ifndef        SLASH
# define  SLASH                 '/'
# endif

# ifndef SHELL
# define SHELL                  "shell"
# endif

# ifndef REG
# define REG                    register
#endif

# ifndef NEEDSYNC
# define NEEDSYNC       FALSE
# endif

# ifndef FILEPERMS
# define FILEPERMS      0666
# endif
# ifndef PRESERVE
define                          PRESERVE
"/usr/libexec/elvispreserve"
# endif

# ifndef CC_COMMAND
# define CC_COMMAND             "cc-c"
# endif

# ifndef MAKE_COMMAND
# define MAKE_COMMAND       "make2
# endif

# ifndef REDIRECT
# define REDIRECT       "2>"
# endif

# ifndef BLKSIZE
# ifndef CRUNCH
# define BLKSIZE        1024
# else
# define BLKSIZE        2048
#  endif
# endif

# ifndef KEYBUFSIZE
# define KEYBUFSIZE     1000
```

```
# endif

# ifndef MAILER
# define MAILER                 "mail"
# endif

ifndef gethome
extern char *gethome();
# endif
# endif
```

CTYPE.H

```
# ifndef_CT_UPPER

# define_CT_UPPER       0x01
# define_CT_LOWER       0x02
# define_CT_SPACE       0x04
# define_CT_DIGIT       0x08
# define_CT_ALNUM       0x10
# define_CT_CNTRL       0x20

# define isalnum(c)             (_ct_ctypes[ UCHAR(c)
] &_CT_ALNUM)
# define isalpha(o)             (_ct_ctypes[UCHAR(c)
] & (_CT-LOWER|_CT-UPPER))
# define isdigit(c) (_ct_ctypes[UCHAR(c) ] &_CT_
DIGITT)
# define islowerte(c)           (_ct_ctypes[UCHAR(c)
] &_CT_LOWER)
# define isspace(c)              (_ct_ctypes[UCHAR(c)
] &_CT_SPACE)
# define isupper(c9             (_ct_ctypes[UCHAR(c)
] &_CT_UPPER)
# define iscntr(c)              (_ct_ctypes[UCHAR(c)
] &_CT_CNTRL)
# define ispunct(c)             (!_ct_ctypes[UCHAR(c)
])

# define isascii(c) (!((c) & 0x80))

# define toupper(c)
        _ct_toupper[UCHAR(c)]
# definetolower(c)              _cttolower[UCHAR(c)]
extern uchar_ct_toupper[],
extern uchar_ct_tolower[],
extern uchar_ct_ctypes[],
extern void_ct_init( ) ;

# endif
```

CURSES.H

```
extern char *tgoto();
extern char *tgetstr();
extern void  tputs().
# if MSDOS
extern int     vmode;
extern void v_up ( ) ;
extern void v_ob ( ) ;
extern void v_es ( ) ;
extern void v_ce ( ) ;
extern void v_cl ( ) ;
extern void v_cd ( ) ;
extern void v_al ( ) ;
extern void v_dl ( ) ;
extern void v_sr ( ) ;
extern void v_move ( ) ;
#endif

extern int              faddch ( ) ;
# define WINDOW         char
```

```
# define TRUE                   1
# define FALSE                  0
# define A_NORMAL       0
# define A_STANDOUT     1
# define A_BOLD                 2
# define A_UNDERLINE 3
# define A_ALTCHARSET           4
# define A_POPOP                5
# define A_VISIBLE      6
# define KBSIZ                  4096
# ifndef NO_FKEY
# ifdef NO_SHIFT_FKEY
# define NFKEYS         10
# else
# ifdef NO_CTRL_FKEY
# define NFKEYS         20
# else
# ifdet NO_ALT_FKEY
# define . NFKEYS       30
# else
# define NFKEYS         40
#   endif
#  endif
# endif
extern char *FKEY[NFKEYS] ;
# endif

extern char *termtype,
extern short ospeed,
# if OSK
extern char PC_ ;
extern char *BC;
# else
extern char PC;
# endif
extern WINDOW           *stdser;
extern WINDOW           kbut[KBSIZ];
extern int      LINES;
extern int      COLS;
extern int      AM;
extern int      PT;
extern char     *VB;
extern char     *UP;
extern char     *SO;
extern char     *SE;
extern char     *US;
extern char     *UE;
extern char     *MD;
extern char     *ME;
extern char     *AS;
extern char     *AE;
# ifndef NO_VISIBLE
extern char     *MV;
# endif
extern char     *CM;
extern char     *CE;
extern char     *CD;
extern char     *AL;
extern char     *DL;
# if OSK
extern char     *SR_;
# else
extern char     *SR;
# endif
extern char     *KS
extern char     *KE;
extern char     *KU;
extern char     *KD;
extern char     *KL;
extern char     *KR;
extern char     *PU;
extern char     *PD;
```

```
extem char      *HM;
extem char      *EN;
extem char      *KI;
extem char      *IM;
extem char      *CL;
extem char      *EI;
extem char      *DC;
extem char      *TI;
extem char      *TE;
# ifndef NO_CURSORSHAPE
extem char      *CQ;
extem char      *CX;
extem char      *CV;
extem char      *CI;
extem char      *CR;
#endif
extem char      *send;
extem char      ERASEKEY;
#ifndef NO_COLOR
extem char      Secolor[];
extem char      SEcolor[];
extem char      UScolor[];
extem char      UEcolor[];
extem char      MDcolor[];
extem char      MEcolor[];
extem char      AScolor[];
extem char      AEcolor[];
# ifndef NO_POPUD
extem char POPUPcolor[];
# endif
# ifndef NO_VISIBLE
extem char  VISIBLEcolor[];
# endif
EXTEM CHAR NORMALCOLOR [];
# endif

# if MSDOS
extem char o_pcbios[1];
# define        CHECKBIOS(x,y)
        (* o_prebios ? (x) : (y))
# define VOIDBIOS (x,y) {if(* o_pebios) {x ;}else
{y;}}
# else
# define        CHECKBIOS (x,y)         (y)
# define VOIDBIOS(x,y) {y;}
# endif


# ifndef
#               define              setcolor(m,a)
CHECKBIOS(bioscolor(m,a), ansicolor(m,a))
# define fixcolor()
VODBIOS(:,tputs(normacolor,1,faddch))
# define quitcolorCHECKBIOS(biosquit(),
ansiquit())
# define do_SE()
VOIDBIOS((vmode=A_NORMAL),
tputs (Secolor,1,faddch))
# define do_US()
VOIDBIOS((vmode=A_UNDERLINE),
tputs(UScolor,1,faddch))
# define do_UE()
VOIDBIOS((vmode=A_NORMAL),
tputs(UEcolor,1,faddch))
# define do_MD()
VOIDBIOS((vmode=A_BOLD),
tputs(MDcolor,1,faddch))
# define do_ME()
VOIDBIOS((vmode=A_NORMAL),
tputs(MEcolor,1,faddch))
# define do_AS()
VOIDBIOS((vmode=A_ALTCHARSET),
tputs(AScolor,1,faddch))
# define do_AE()
```

```
VOIDBIOS((vmode=A_NORMAL),
tputs(AEcolor,1,faddch))
# define do_POPUP()
VOIDBIOS((vmode=A_POPUP),
tputs(POPUPcolor,1,faddch))
# define do_VISIBLE()
VOIDBIOS((vmode=A_VISIBLE),
tputs(VISIBLEcolor,1,faddch))
# else
# define do_SO()
VOIDBIOS((vmode=A_STANDOUT),
tputs(SOcolor,1,faddch))
# define do_SE()
VOIDBIOS((vmode=A_NORMAL),
tputs(SEcolor,1,faddch))
# define do_US()
VOIDBIOS((vmode=A_UNDERLINE),
tputs(UScolor,1,faddch))
# define do_UE()
VOIDBIOS((vmode=A_NORMAL),
tputs(UEcolor,1,faddch))
# define do_MD()
VOIDBIOS((vmode=A_BOLD),
tputs(MDcolor,1,faddch))
# define do_ME()
VOIDBIOS((vmode=A_NORMAL),
tputs(MEcolor,1,faddch))
# define do_AS()
VOIDBIOS((vmode=A_ALTCHARSET),
tputs(AScolor,1,faddch))
# define do_AE()
VOIDBIOS((vmode=A_NORMAL),
tputs(AEcolor,1,faddch))
# define do_POPUP()
VOIDBIOS((vmode=A_POPUP),
tputs(SOcolor,1,faddch))
# define do_VISIBLE()
VOIDBIOS((vmode=A_VISIBLE),
tputs(MVcolor,1,faddch))
# endif
# define do_VB()
VOIDBIOS((..tputs(VB,1faddch))
# define do_UP()
VOIDBIOS((v_up(),tputs(UP,1,faddch))
# undef do_CM
# define do_CE()
VOIDBIOS((v_ce(),tputs(CE,1,faddch))
# define do_CD()
VOIDBIOS((v_cd(),tputs(CD,1,faddch))
# define do_AL()
VOIDBIOS((v_al(),tputs(AL,LINES,faddch))
# define do_DL()
VOIDBIOS((v_dl(),tputs(DL,LINES,faddch))
# if OSK
# define do_SR()
VOIDBIOS((v_sr(),tputs(SR_,1,faddch))
# else
# define do_SR()
VOIDBIOS((v_sr(),tputs(SR,1,faddch))
# endif
# define do_KS()
VOIDBIOS((v_up(),tputs(KS,1,faddch))
# define do_KE()
VOIDBIOS((v_up(),tputs(KE,1,faddch))
# define do_IM()
VOIDBIOS(..tputs(IM,1,faddch))
# define do_IC()
VOIDBIOS(..tputs(IC,1,faddch))
# define do_EI()
VOIDBIOS(..tputs(EI,1,faddch))
# define do_DC()
VOIDBIOS(..tputs(DC,COLS,faddch))
```

```
# define do_TI()
VOIDBIOS(,,(void)ttywrite(TI,(unsignedstrlen(TI)
))
# define do_TE()
VOIDBIOS(,,(void)ttywrite(TE,(unsignedstrlen(T
E))
# ifndef NO_CURSORSHAPE
# define do_CQ()
VOIDBIOS(v_cs(),tputs(CQ,1,faddch))

# define do_CX()
VOIDBIOS(v_cs(),tputs(CX,1,faddch))
# define do_CV()
VOIDBIOS(v_cs(),tputs(CV,1,faddch))
# define do_CI()
VOIDBIOS(v_cb(),tputs(CI,1,faddch))
# define do_CR()
VOIDBIOS(v_cb(),tputs(CR,1,faddch))
# endif
# ifndef NO_COLOR
# define do_aend()
VOIDBIOS(vmode=A_NORMAL),endcolor())
# else
# define do_aend()
VOIDBIOS(vmode=A_NORMAL),tputs(aend,1,fa
ddch))
# endif
# define has_AM          CHECKBIOS(1,AM)
# define has_PT          CHECKBIOS(0,PT)
# define has_VB  CHECKBIOS((char *)0,VB)
# define has_UP  CHECKBIOS((char *)1,UP)
# define has_SO  CHECKBIOS((char )1,(*SO))
# define has_SE  CHECKBIOS((char )1,(*SE))
# define has_US  CHECKBIOS((char )1, (*US))
# define has_UE  CHECKBIOS((char )1, (*UE))
# define has_MD CHECKBIOS((char )1, (*MD))
# define has_ME CHECKBIOS((char )1, (*ME))
# define has_AS  CHECKBIOS((char )1, (*AS))
# define has_AE  CHECKBIOS((char )1, (*AE))
# undef  has_CM
# define has_CB  CHECKBIOS(1,0)
# define has_CS  CHECKBIOS(1,0)
# define has_CE  CHECKBIOS((char *)1,CE)
# define has_CD  CHECKBIOS((char *)1,CD)
# define has_AL  CHECKBIOS((char *)1,AL)
# define has_DL  CHECKBIOS((char *)1,DL)
# if OSK
# define has_SR  CHECKBIOS((char *)1,SR_)
# else
# define has_SR  CHECKBIOS((char *)1,SR)
# endif
# define has_KS  CHECKBIOS((char)1,(*KS))
# define has_KE  CHECKBIOS((char)1,(*KE))
# define has_KU                  KU
# define has_KD                  KD
# define has_KL                  KL
# define has_KR                  KR
# define has_HM                  HM
# define has_EN                  EN
# define has_PU                  PU
# define has_PD                  PD
# define has_KI                  KI
# define has_IM  CHECKBIOS((char )0, (*IM))
# define has_IC  CHECKBIOS((char )0, (*IC))
# define has_EI  CHECKBIOS((char )0, (*EI))
# define has_DC  CHECKBIOS((char *)0, DC)
# define has_TI  CHECKBIOS((char )0, (*TI))
# define has_TE  CHECKBIOS((char )0, (*TE))
# ifndef NO_CURSORSHAPE
# define has_CQ  CHECKBIOS((char )1, (*CQ))
# endif
# ifdef lint
# define_addCR  VOIDBIOS(,,(stdscr[-1]=='\n'?
```

```
qaddch('\r') (stdscr[-1]='\n')))
# else
# if OSK
# define_addCR  VOIDBIOS(:,(stdscr[-1]=='\n'?
qaddch('\1')=stdscr[-1] )))
# else
# define_addCR  VOIDBIOS(:,(stdscr[-1]=='\n'?
qaddch('\r') 0))
# endif
# endif
# ifdef AZTEC_C
# define qaddch(ch)
CHECKBIOS(v_put(ch),(*stdscr=(ch),*stdscr++))
# else
CHECKBIOS(v_put(ch),(*stdscr++=(ch))
# endif
# if OSK
# define addch(ch)
if (qaddch(ch)=='\n')qaddch('\l'), else
# else
if (qaddch(ch)=='\n')qaddch('\r'), else
# endif
extern void initscr(),
extern void endwin(),
extern void suspend_curses(),
extern void resume_curses(),
extern void attrset(),
extern void insch(),
extern void qaddstr(),
extern void wrefresh(),
extern void wrqrefresh(),
# define addstr(str) {qaddstr(str),_addCR,}
# define move(y,x)
VOIDVIOS(v_move(x,y),tputs(tgoto(CM,x,y),1,fa
ddch))
# define mvaddch(y,x,ch)  {move(y,x),addch(ch),}
# define refresh() VOIDBIOS(,, wrefresh())
# define standout()        do_SO()

# define standend()        do_SE()
# define clrtoeol()        do_CE()
# define clrtobot()        do_CD()
# define insertln()do_AL()
# define deletein()        do_DL()
# define delch()           do_DC()
# define scrollok(w,b)
# define raw()
# define echo()
# define cbreak()
# define noraw()
# define noecho()
define nobreak()
```

REGEXP.H

```
# define NSUBEXP 10

typedef struct regexp {
        char    *startp[NSUBEXP],
        char    *endp[NSUBEXP],
        int     minlen,
        char    first,
        char    bol,
        char    program[1],
} regexp,
extern regexp *regcomp(),
extern int regexec(),
extern int regsub(),
extern void regerr(),
```

ENTRADA.C

```
# include "config.h"
```

```
# include "ctype.h"
# include "vi.h"

# ifndef NO_DIGRAPH
static struct_DIG
{
        struct_DIG *next,
        char    key1,
        char    key2,
        char    dig,
        char    save,
} *digs;
char digraph(key1,key2)
        int     key1,
        int     key2,
{
        int     newkey,
        REGstruct_DIG  *dp,

        if(!*o_digraph
        {
                return key2,
        }
        newkey=key2;
        if(key1>key2)
        {
                key2=key1,
                key1=newkey,
        }
        for (dp=digs,
 dp && (dp->key1 !=key1 || dp->key2 !=key2),
        dp =dp->next)
        {
        }
        if(!dp)
        {
                return newkey,
        }
                return dp->dig,
}
void do_digraph(bang,extra)
        int     bang,
        char    extra[],
{
        int     dig,
        REG struct_DIG          *dp,
        struct_DIG *prev,
        static int user_defined=FALSE,
        char    listabuf[8],
        if (!extra)
        {
                user_defined=TRUE;
                return,
        }
        if (*extra<' ')
        {
listbuf[0]=listbuf[1]=listbuf[2]=listbuf[5]=' ';
        listbuf[7]='\0';
        for (dig=0,dp=digs,dp;dp=dp->next)
        {
                if(dp->save || bang)
                {
                        dig +=7,
                        if (dig>=COLS)
                        {
        addch('\n'),
        exrefresh(),
                        dig=7;
                        }
                        listbuf[3]=dp->key1,
                        listbuf[4]=dp->key2,
```

```
                        listbuf[6]=dp->dig,
                        qaddstr(listbuf),
                        }
                }
                addch('\n'),
                exrefresh(),
                return;
        }
        {
        dp=(struct_DIG*)malloc(sizeof*dp),
                if (!dp)
                {
        msg("Tabla de compuestos llena"),
                }
                if (prev)
                        prev->next=dp,
                else
                        digs=dp,
                dp->next=(struct_DIG *)0,
        }
        dp->key1=extra[0],
        dp->key2=extra[1],
        dp->dig=dig,
        dp->save=user_defined,
}
# ifndef NO_MKEXRC
void savedigs(fd)
        int     fd,
{
        static char buf[]="digraph! XX Y\n",
        REG struct_DIG          *dp,
        for (dp=digs,dp,dp=dp->next)
        {
                if (dp->save)
                {
                                buf[9]=dp->key1,
                                buf[10]=dp->key1,
                                buf[12]=dp->key1,
                        write(fd,buf,(unsigned)14),
                        }
        }
}
# endif
# endif
MARK input(from,to,when,delta)
        MARK    form,
        MARK    TO,
        int     when,
        int     delta,
{
        char    key[2],
        char    *build,
        char    *scan,
        MARK m,
# ifndef NO_EXTENSIONS
        int     quit=FALSE,
        int     inchg,
# endif
# ifndef DEBUG
        if (from>to)
        {
msg("ERROR input(%ld %d,%ld,%d)",
markeline(from),markidx(from),markeline(to),mark
idx(to)),
                return MARK_UNSET,
        }
# endif
        key[1]=0,
        if (from != to)
        {
                cut(from,to),
```

```
        }
        if (doingdot)
        {
                change Text
                {
                        if (from != to)
                        {
                                delete(from to);
                        }
                        cutname('.');
        cursor=paste(from,FALSE,TRUE)+IL;
                }
        }
        else
        {
# ifndef NO_EXTENSIONS
                incing=TRUE;
# endif
if (from != to && markeline(from)==makeline(to))
                {
                change(to-1,to,"$");
                }
                else
                {
                        if (from != to)
                        {
                                delete(from,to);
                        }
                        to=from;
                }
                cursor=from;
                m=cursor+MARK_LINE(delta);
                if (delta != 0 && *o_autoindent
&& markidx(m))=0
&& markline(m)>= IL && markline(m) <= nlines)
                {
                        pfech(markline(cursor));
                        if (plen==0)
                        {
                        pfetch(markline(m));
                for (scan=prext,build=tmpblk.c,
                *scan==' '||*scan=='\t');
                        {
                        *build='\0';
        add(cursor,tmpblk.c);
                cursor+=(int)(buid-tmpblk.c);
                        if (cursor > to)
                to=cursor;
                                }
                        }
                }
                for(;;)
                {
                        redraw(cursor,TRUE);
# ifdef EBUG2
msg("cursor=%id,%d,a=%ld,%d",makeline(cursor
),markidx(cursor),markline(to),markidx(to));
# endif
# ifndef NO_ABBR
                        pfech(markline(cursor));
                        build=ptext;
                if (pline==markline(from))
                        build+=markidx(from);
        for(scan=plext+markidx(cursor),-
scan>=buid && !isspace(*scan);)
                        {
                        }
                        scan++;
        key[0]=getabkey(when,scan,(int)(ptext+m
arkidx(cursor)-scan);
# else
                key[0]=getkey(when);
# endif
```

```
# ifndef NO_VISIBLE
                if (key[0] !=ctrl('O') && V_from
!=MARK_UNSET)
                {
                        msg("no puedo
modificar texto durante una selección"),
                                beep();
                                continue;
                        }
# endif
# ifndef NO_EXTENSIONS
                        if (key[0]==ctrl('O'))
                        {
                                if (inchg)
                                {
                                if (cursor < to)
                                {
        delete(cursor to);
        redraw(cursor,TRUE);
                                }
                                afterdo();
                                inchg=false,
                        }
                }
                else if (key[0] !=ctrl('['))
                {
                        if (!inchg)
                        {
        beforedo(FALSE),
                        inchg=TRUE,
                        }
                }
# endif
# ifndef CRUNCH
                if (*o_wrapmargin)
                {
                        pfech(makline(cursor)),
                        if (plen==idx2col(cursor,ptext,TRUE))
&& plen > COLS- (*o_wrapmargin & 0xff))
                                {
                                build=tmpblk.c;
                                *build++='\n';
                                if (*o_autoindent)
                                {
for (scan=ptext; *scan == ' ' || *scan == '\t';)
        {
                *build++=*scan++;
        }
                                }
                                *build = '\0';
                                scan=ptext + plen;
cursor & ~(BLKSIZE-1); m=(ptext<scan)
                                while (ptext<scan)
                                {
                        scan-;
                        if (*scan != ' ' && *scan != '\t')
                        continue;
                change(m+(int)(scan-ptext),m+(int)(scan-
ptext)+1,tmpblk.c);
                          cursor=(cursor & ~(BLKSIZE-
1)),
                        cursor+=BLKSIZE;
                        cursor+=strlen(tmpblk.c)-1,
                        cursor+=plen-(int)(scan-ptext)-1,
                        pfech(makline(m));
                        scan=ptext+plen;
                        while (ptext<scan)
                        {
                                scan-,
                        if (*scan!=' ' && *scan !='\t')
                                break,
                        }
```

```
        delete(m+(int)(scan-ptext)+1,m+plen),
                break;
                        }
                }
# endif
                switch(*key)
# ifndef NO_EXTENSIONS
                case ctrl('O'),
                        *key =getkey(0),
                        switch (*key)
                        {
                case 'h': m=m_left(cursor,0L),
                break;
                case 'j':
         case 'k',mv=m _updnto(cursor,0L,*key),
                break;
                case 'l': m=cursor+1;
                break;
                case 'B':
         case 'b': m=m_bword(cursor,0L,*key);
                break;
                case 'W':
        case 'w': m=m_fword(cursor,0L,*key,'\0');
                break;
                case w:                 m=
                case"$"
                break,
                case ctr('B')
                case ctrl('F')
m=m_scroll(cursor, OL, *key), break,
                case 'x'
ifndef NOVISIBLE
                if(V_from)
                                        beep(),
                else
#endif
                        Change Text
                        {

        m=v_xchar(cursor,OL,'x');
                }
                break
from= cursor                    case ii:    m0to=
        when=WHEN_VIINP+WHEN_VIREP-
when;
                                        break;
                                case"K".
        pfetch(markline(cursor);
        changes++;
        ptext[markidx(cursor)]=0;
                                        for(scan
=ptext+markidx(cursor)-1;
                                        scan
>=ptext && isalnum(*scan);
                                        scan-
)
                                        {
                                        }
                                        scan++;
                                        m=
(*scan ? v_keyword(scan,cursor,OL);
                                        break;
# ifndef NO_VISIBLE
                                case 'V':
                                case 'V':
                                        if
(V_from)
        V_from=MARK_UNSET;
                                        else
        V_from=cursor
                                        m=from
=to=cursor,
```

```
        V_linemd=(*key=='V'),
                                        break;
                                case 'd':
                                case 'y':
                                case '\\':
                                        if
(IV_from)
                                        {
        mgs("Debe marcarse el texto antes"),
        beep(),
        break;
                                        }
        from= V_from;
        to= V_from;
                                        }
                                        else
                                        {
        from=cursor;
        to= V_from,
                                        }
=MARK_UNSET,
                                        if
(V-linemd)
                                        {
        from &=(BLKSIZE-1),
        to<=(BLKSIZE-1),
                                        }
                                        else
                                        {
        pfetch(markline(to),
        if (markidx)==plen)
                to<=(BLKSIZE-1),
                                        }
                                        to++,
                                        switch
(*key)
                                                {
                                                case 'y':
cut(from,to);
break,
                                                caso 'd':
        Change Text
        {
                cut(fro,to),
                delete(from,to).
        }
        cursor=from;
        break;
#ifndef NO_POPUD
                                                case\\;
        Change Text
        {
                cursor=v_popup(from,to),
        }
        break;
#endif
                                                }
                                                m=from
= to=cursor;
                                                break;
                                        case 'p':
                                        case 'P':
                                                V_from
=MARK_UNSET:
        Change Text
                                                        {
        m=paste(cursor,(*key== p'),FALSE,
                                                        }
                                                        break,
# endif
                                        defaul          m=
```

```
MARK_UNSET
                                        )
                                        if(m<=
cursor)
                                {
                                        m=
adjmove(cursor,m(*key    =='j'    ||    *key=='k'?
NCOL|FINL:FINL));
                                        if
(plen
&& (*key == 'S'|| (*key == T&& m <=cursor)))
                                        {
        m++;
                                        }
                                }
                                if(m==
MARK_UNSET)
                        {
                                beep();
                        }
                else
                        {

        from=to
=cursor=m;
                        }
                        break;
                case ctrl('Z')
                        if (getkey(0)==
ctrl('Z'))
                        {
                                quit=
TRUE;
                                goto
BreakBreak;
                        }
                        break;
#endif
                case ctrl('T
                        if(*o_autoindent)
                        {
        pfetch(markline(cursor));

        for(scan
= ptext;isspace(*scan);scan++
                                {
                                }
                                if(scan>
ptext && !*scan)
                                {
        cursor &=(BLKSIZE-1L),
        if (to<cursor+plen)
        {
                to=cursor + plen;
        }
                                }
                        }
                        goto BreakBreak;
                case ctrl('U'):
                        if (markline8cursor)
==markline(from))
                        {
                                cursor=
from;
                        }
                        else
                        {
                                cursor
&=~(BLKSIZE-1),
                        }
                        break;
```

```
                case ctrl('D'),
                case ctrl('T'):
                                if(to> cursor)
                {
delee(tusor,to),
                        {
                mark[27]=cursor;
end_shif(cursor,cursor,*key==ctrsl(D)                ?
CMD_SHISFTL : CMD_SHIFTR, TRUE, "");
                        if (mark[27])
                        {
                        cursor=mark[27];
                        }
                        else
                        {
                cursor=m_front(cursor,0L),
                        }
                        to =cursor;
                        break;
                case '\b';
                        if (cursor<=form)
                        {
                                beep();
                        }
                        else if
(markidx(cursor)==0)
                        {
                        cursor=BLKSIZE,
        pfetch(markline(cursor));
                        cursor+=plen,
                        }
                        else
                        {
                                cuirsor=,
                        }
                        break,
                case ctrl('W'),
                        m=m_bword(cursor,IL,'b');
        if (markline(m))==markline(cursor) && m
>= from)
                                {
                                cursor=m;
                                if (from>cursor)
                                {
                        from=cursor;
                                }
                        }
                        else
                        {
                                beep();
                        }
                        break,
        case '\n',
# if OSK
        case '\l',
# else
        case '\r',
# endif
                                build=tmpblk_c,
                                *build++='\n';
                                if (*o_autoindent)
                                {
        pfetch(markline(cursor)),
                        for (scan=ptext, *scan == ' ' ||
*scan == '\t';)
                                        {
                                *build++ = *scan++;
                                        }
                                                if
((int)(scan-ptext)>=msrkidx(cursor) && plen > 0)
                                                {
        to= cursor & =~(BLKSIZE-1),
```

```
        delete(cursor,cursor+(int)(scan-ptext));
                                }
# if 0
                        if (to>= cursor)
                        {
for (scan = ptext+markidx (cursor), to = cursor,
*scan == ' ' ||*scan == '\t'; scan++, to++)
        {
        }
                                }
# endif
                        }
                        *build=0;
                        if (cursor>= to &&
when !=WHEN_VIREP)
                        {
        add(cursor,tmpblk,e);
                        }
                        else
                        {
        change (cursor,to tmpblk,c);
                        }
                        redraw(cursor,TRUE);
                to=cursor = (cursor &
~(BLKSIZE-1))
        + BLKSIZE
        +(int)(build-tmpblk.c)-1;
                                break;
                        case ctrl('A');
                        case ctrl('P');
                        if (cursor>to)
                        {
        delete(cursor,to);
                        }
                m=paste(cursor,FALSE,TRUE);
                if(m!=MARK_UNSET)
                        {
                to=cursor=m+1L;
                        }
                        break;
                case ctr('V');
                        if (cursor>= to &&
when !=WHEN_VIREP)
                        {
                add(cursor,"^");
                        }
                        else
                        {
                change(cursor,to,"^");
                                to=cursor+1;
                        }
        redraw(cursor,TRUE);
                        *key=getkey(0);
                        if(*key=='\n')
                        {
# if OSK
                                *key='\t';
# else
                                *key='\r';
# endif
                        }
                change(cursor,cursor+1,key);
                        cursor++;
                        if (cursor>yo)
                        {
                                to=cursor;
                        }
                                break;
                case ctrl('L');
                case ctrl('R');
        redraw(MARK_UNSET,FALSE);
                                break;
```

```
                default:
                        if (cursor >= to
&& when != WHEN_VIREP)
                        {
        add(cursor,key);
                        cursor++;
                        to cursor;
                        }
                        else
                        {
        pfetch(markline(cursor));
                        if
(markidx(cursor)=plen)
                        {
                add(cursor,key);
                        }
                        else
                        {
# ifndef NO_DIGRAPH
*key = digraph(ptext[markidx(cursor)],*key);
# endif
        change(cursor,cursor+1,key);
                                }
                                cursor++;
                }
#  ifndef NO_SHOWMATCH
        if (*o_showmatch && strchr(")}]",*key))
                {
        redraw(cursor,TRUE);
                m=m_match(cursor-1,0L);
                if (markline(m)>=topline  &&
markline(m)<=botline)
                        {
                redraw(m,TRUE);
                refresh();
                sleep(1);
                                }
                        }
break;
                if (cursor<to)
                {
# ifndef NO_EXTENSIONS
                        if (!inchg)
                        {
                        beforedo(FALSE);
                        inchg=TRUE;
                        }
# endif
                        delete(cursor,to);
                }
        }
        if (!doingdot)
        {
                blksync();
                cutname('.');
                cut(from,cursor);
        }
        if (markidx(cursor)! =0)
        {
                cursor-;
        }
        redraw(cursor,FALSE);
#ifndef NO_EXTENSIONS
        if (quit)
        {
                abortdo();
                cursor=v_xit(cursor,0L,'Z');
        }
# endif
                rptlines=0L;
                return cursor;
```

```
}

VI.H

# include<crmo.h>
# if TOS
# ifndef_GNUC_
#define ENOENT(-AEFILNF)
# endif
# endif

# if TOS || VMS
# include <ctypes.h>
# define O_RDONLY      0
# define O_WRONLY      1
# define O_RDWR        2
# ifndef _GNUC_
# define S_UDIR        S_IFDIR
# endif
# else
# if OSK
# include <modes.h>
# define O_RDONLY      S_IREAD
#define O_WRONLY      S_IWRITE
# define O_RDWR    (S_IREAD | S_IWRITE)
# define ENOENT        E_PNNF
# define  sprintf      Sprintf
# else
# if AMIGA
# include <sys/type.h>
# endif
# if COF_286
# include <sys/fcntl.h>
# else
# include <fcntl.h>
# endif
# endif
# endif

# ifndef O_BINARY
# define O_BINARY      0
# endif

# include 'curses.h'

# include <signal.h>
# ifdef_STDC_
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include <stdarg.h>
# if ANY_UNIX
# include <unistd.h>
# include <sys/wait.h>
# endif
# endif

# ifdef REGEX
# define SE_MAX                30
# endif
# define INFINITY          2000000001L
# define LONGKEY               10
# ifndef MAXRCLEN
# define MAXRCLEN              1000
# endif

# define MAXBLKS
        (BLKSIZE/sizeof(unsigned short))
typedef union
{
                char      c[BLKSIZE];
```

```
                unsigned short    n[MAXBLKS];
                }
                BLK;
extern BLK hdr;
extern BLK *blkget P_((int));
extern BLK *blkadd P_((int));

extern struct_viflags
{
        short file;
}
        viflags
# define NEWFILE                0X0001
# define READONLY       0X0002

# define HADNUL                 0X0004
# define MODIFIED       0X0008
# define NOFILE                 0X0010
# define ADDEDNL                0X0020
# define HADBS                  0X0040
# define UNDOABLE       0X0080
# define NOTEDITED      0X0100

# define setflag(x,y)           viflags x!=y
# define ctrflag(x,y)           viflags x&~=y
# define tstflag(x,y)           (viflags x& y)
# define initflag() viflags file=0;
extern o_autoindet[1];
extern o_autoprint[1];
extern o_autotab[1];
extern o_autowrite[1];
extern o_columns[3];
extern o_directory[30];
extern o_edcompatible[1];
extern o_equalprg[80];
extern o_errorbells[1];
extern o_exrefresh[1];
extern o_ignorecase[1];
extern o_keytime[3];
extern o_keywordprg[80];
extern o_lines[3];
extern o_list[1];
extern o_number[1];
extern o_readonly[1];
extern o_remap[1];
extern o_report[3];
extern o_scroll[3];
extern o_shell[60];
extern o_shiftwidth[3];
extern o_sydescroll[3];
extern o_sync[1];
extern o_tabstop[3];
extern o_term[30];
extern o_flash[1];
extern o_wam[1];
extern o_wrapecan[1];

# ifndef CRUNCH
extern char  o_beautify[1];
extern char  o_exrc[1];
extern char  o_mesg[1];
extern char  o_more[1];
extern char  o_nearscroll[3];
extern char  o_novice[1];
extern char  o_prompt[1];
extern char  o_taglenght[3];
extern char  o_tags[256];
extern char  o_terse[1];
extern char  o_window[3];
```

```
extem char  o_wrapmargin[3],
extem char  o_writeany[1],
#endif

#ifndef NO_ERRLIST
exterm char o_cc[30],
extem char  o_make[30],
#endif

ifndef NO_CHARATIR
extem char o_charatt[1],
#endif

#ifndef NO_DIGRAPH
extem char o_digraph[1],
extem char o_flipoase[80],
#endif

#ifndef NO_SENTENCE
extem char o_hideformat[1],
#endif

#ifndef NO_EXTENSIONS
extem char o_imputmodo[1];
extem char o_ruter[1],
#endif

#ifndef NO_MAGIC
extem char o_magic[1],
#endif
```

```
#ifndef NO_MODELINES
extern char o_modelines[1];
#endif

#ifndef NO_SENTENCE
extern char o_paragraphs[30];
extern char o_sections[30];
#endif

#if MSDOS
extern char o_pobios[1];
#endif

#ifndef NO_SHOWMATCH
extern char o_showmatch[1];
#endif

#ifndef NO_SHOWMODE
extern char o_amd[1];
#endif

#ifndef NO_TAGSTACK
extern char o_tagstack[1];
#endif

extern char U_text[BLKSIZE];
extern long U_line;

typedef long MARK;
#define markline(x)     (long)(x)/BLKSIZE)
#define MARK_UNSET ((mark)0)
#define                         MARK_FIRST
((MARK)(nlines*BLKSIZE))
#define MARK_LAST ((nlines+1)*BLKSIZE-1))
#define                         MARK_EOF
((MARK)((nlines+1)*BLKSIZE)
#define                 MARK_AT_LINE(x)
((MARK)(x)*BLKSIZE)

#define NMARKS          29
extern MARK     mark[NMARKS];
extern MARK     cursor;

extern long origteme
extern char origname[256];
extern prevorig[256];
extern prevline;

extern int tmpfd;
extern int ttmpnum;
extern long lnum[MAXBLKS];
extern long nlines;
extern char  args[BLKSIZE];
extern int argno;
extern int nargs;
extern long changes;
extern int significant;
extern int exitcode;
extern BLK tmpblk;
extern long topline;
extern int leftcol;

#define             botline   (toplines+LINES-2)
#define             rightcol(leftcol+COLS-
(*o_number ?9 1))
extern int physcol;
extern int phyrow;
extern int exwrote;
extern int doingglobal;
extern long rptlines;
extern char *rptlabel;
extern char *fetchline P_((long));
```

```
extern char *parseptrn P_((REG char*)),
extern MARK     paste P_((MARK,int,int)),
extern char *wildcard P_((char*)),
extern MARK     input
P_((MARK,MARK,int,int)),
extern char *linespec P_((REG char *,MARK*)),
#define             ctrl(ch) ((ch)&037)
#ifndef NO_RECYCLE
extern long allocate P_((void)),
#endif
extern SIGTYPE          trapint P_((int)),
extern SIGTYPE          deathtrap P_((int)),
extern void blkdirty P_((BLK *)),
extern void blksync P_((void)),
extern void blkinit P_((void)),
extern void beep P_((void)),
extern void exrefresh P_((void)),
#ifdef              __STDC__
extern void msg (),
#endif
```

CONCLUSIONES.

El editor de pantalla que se desarrolló en el presente proyecto, será muy familiar para los usuarios de Unix. La interfaz que presenta al usuario es la que usualmente se encuentra en este ambiente de sistema operativo; aun así, la intención original de hacerlo tan amigable como fuera posible, se ,mantuvo en mente al escribir el código. Bajo cierta óptica, puede considerarse como una limitante el hecho de que su operación no sea tan accesible como la de lo editores comerciales que operan en sistemas personales, aunque también es cierto que su entorno es y seguirá siendo enteramente distinto.

Para el programador que realice una revisión al código actual, el manejo de las áreas de corte resultará particularmente interesante, ya que el trabajo aporta un esquema que es aplicable a otro tipo de proyectos que demanden de un trabajo similar de memoria; tal es el caso de algunos algoritmos de búsqueda para bases de datos o bien en otros métodos de inteligencia artificial. También destaca la sección que se escribió para la evaluación de expresiones regulares, la cual puede usarse en un sinnúmero de programas que requieren de rutinas para el manejo de este tipo de cadenas. Sin embargo, la principal virtud del proyecto, es que ilustra las múltiples facetas de la programación bajo Unix; desde el manejo de memoria hasta la creación y administración de los procesos que interactúan directamente con éste último.

Cabe señalar que aún existe un campo potencial de mejoras al producto de este

trabajo, y que la mayoría de ellas pueden orientarse a la compatibilidad con otros editores muy difundidos en el ambiente Unix, como es emacs. Los principales problemas que podrán encontrarse ante tales modificaciones se limitarán a la ampliación de funciones y a un manejo ligeramente distinto de las expresiones regulares, pero la funcionalidad para el manejo de la pantalla y del teclado resultará prácticamente idéntica.

APÉNDICE.

A continuación se presenta una breve lista de los comandos más utilizados en el entorno operativo Unix.

COMANDOS	ACCIÓN QUE REALIZA
cancel request	Para cancelar una solicitud de impresión
cat filenames	imprimir contenidos de los archivos nombrados.
cd	Para ir al directorio hogar
cd patname	Para ir a un determinado directorio de trabajo
cmp file1 file2	imprimir localización de la primera diferencia
cp file 1 file 2	copiar file1 en file2, sustituir el file2 anterior si existe.
diff file1 file2	imprimir todas las diferencias entre archivos.
ed filename	editar archivo nombrado.
grep pattern filenames	imprimir líneas que coincidan con *pattern*.
grep -v pattern files.	imprimir líneas que no coincidan con *pattern*
kill PID	Para detener el proceso de un PID.
lp files	Para enviar archivos a una impresora por omisión.
lpq	Verificar las solicitudes que están en la línea de espera de la impresora **lpr.**
lpr files	Para enviar archivos a una impresora por omisión.
lprm request	Para cancelar la solicitud de impresión. **lpr.**
lpstat	Para verificar las solicitudes que están en una lista de

	espera de impresión **lpstat**.
ls	lista los nombres de todos los archivos en el directorio actual.
ls filenames	lista sólo los archivos nombrados.
ls -l	lista larga: más información; también **ls - lt**
ls -r	listar en orden inverso; también **- rt, -rlt, etc.**
ls -t	lista por orden de tiempo, primero el más reciente.
ls -u	listar por tiempo el último en usarse; también **ls -lu, ls - lut.**
mail	Para leer su propio correo.
mail user	Para enviar correo al usuario.
man command	Para ver la página del manual correspondiente al comando.
Mkdir path	Para crear un nuevo directorio mediante el nombre de la ruta.
more files	Para presentar pantalla por pantalla cada archivo.
mv file 1 file 2	mover file1 a file 2, sustituir el file2 anterior si existe.
pg files	Para presentar pantalla por pantalla el archivo.
pr filenames	imprimir contenidos con encabezado, 66 líneas por página.
pr -m filenames	imprimir lado a lado archivos nombrados (columnas múltiples)
pr -n filenames	imprimir en *n* columnas.
ps	Para listar sus procesos y sus respectivos PID.
pwd	Para imprimir el nombre del directorio de trabajo actual.

rm filenames	suprimir irrevocablemente los archivos nombrados.
rmdir patname	Para eliminar un directorio que está vacío usando el nombre de la ruta.
sort filenames	clasificar alfabéticamente los archivos por renglón.
tail filename	imprimir los últimos 10 renglones de archivo.
tail +n filename	comenzar a imprimir archivo en el renglón *n* .
tail -n filename	imprimir los últimos *n* renglones de archivo.
wc filenames	contar renglones, palabras y caracteres en cada archivo.
wc - l filenames	cortar renglones de cada archivo.
who	Para listar a los usuarios que están actualmente en el sistema.
who am y	Para presentar el listado correspondiente a la sesión que esté realizando.

BIBLIOGRAFÍA.

SCHILDT, HERBERT. *"Born to code in C"*. Osborne MCGraw-Hill, Estado Unidos, 1989.

KERNIGHAM, BRIAN W. Y RICHIE, DENNIS M. *"El lenguaje de programación C"*. Prentice Hall Hispanoamericana S. A. México D. F. 1986.

KERNIGHAM, BRIAN W. Y PIKE, ROB. *"El entorno de programación Unix"*. Prentice Hall Hispanoamericana S. A. México D. F., 1989.

FIEDLER, DAVID Y HUNTER, BRUCE H. *"Unix System Administration"*. Hayden Books. Estados Unidos, 1990.

HEWLETT PACKARD. *"HP-UX Reference. Release 9.0"*. Vol. 1. Hewlett Packard. Estado Unidos, 1992.

HEWLETT PACKARD. *"HP-UX Reference. Release 9.0"*. Vol. 3. Hewlett Packard. Estado Unidos, 1992.

O'REILLY & ASSOCIATES INC. *"Lernig the unix operting system"*. Prentice Hall. Estado Unidos. 1993.