Sistema operativo UNIX

Antecedentes :

UNIX nació como proyecto universitario y ha llegado a ser uno de los S.O. más empleados actualmente (cada vez más) .

Hace 5 años el 80% de los ordenadores profesionales estaban trabajando prácticamente bajo UNIX.

El UNIX que vamos a utilizar es el UNIX system V release 4 ; el cual incorpora gran variedad de novedades.

La evolución de UNIX continua hasta la estandarización que en teoría nación con el OSF/ 1 (95/96).

Existen diferentes versiones de UNIX : IBM AIX

HP HPUX ...etc.

Estructura :

HARDWARE

NUCLEO

SHELL

APPS. DE USUARIO

Shell Interfaz de comandos.

Recibe ordenes de usuario para el sistema operativo.

Interactuaremos con el Shell.

Es un leguaje de programación lo suficientemente potente para que realicemos operaciones necesarias.

Núcleo Parte del S.O. que interactúa con el Hardware de la máquina recibe llamadas al sistema en ordenes del S.O. Cuando se comunica alguien con el Núcleo realmente lo hace con los procesos que están en espera (o no) en el núcleo.

SISTEMA DE ARCHIVOS

El concepto de archivo es básico en la organización de la información en UNIX. Todo lo que puede tratar con información es un archivo.

Archivo será también una impresora, la pantalla...etc.

UNIX es un sistema dirigido a archivos.

UNIX se basa en tres puntos :

- *Simplicidad*
- *Generalidad*
- *Extensibilidad*

Cualquier tarea que se realice en UNIX está formada por la combinación de componentes simples.

Todas las componentes del sistema tienen un funcionamiento muy global .

Es un gran conjunto de elementos pequeños ; cada una de las tareas hace una función muy pequeña.

Es necesario que todas las ordenes formen una cosas global.

Las herramientas son muy genéricas.

Para programar en UNIX hay que prepararles en muchas partes pequeñas.

HISTORIA UNIX

Nació en 1960 en un sistema operativo llamado MULTICS. Este fue diseñado como un sistema interactivo multiusuario que permitía acceder a la misma información a muchos usuarios .

En 1969 Kemp Thompson continuó desarrollando MULTICS para una máquina llamada PDP−7 hasta obtener un sistema llamado UNICS que incorporaba todas las características del UNIX actual (archivo filosofía de funcionamiento).

El S.O. UNICS continuó evolucionando en máquinas más potentes dando lugar en 1970/71 al conocido UNIX.

A partir de 1970 se empezó a hablar de él. Solo cabe destacar dos sucesos ; en 1973 se unió con el lenguaje C, y se rescribió en C ; pero esta unión no se hizo solo para rescribirse en C , sino que el C se creó para rescribir UNIX de hecho hace que la sintaxis de UNIX y C sean casi hermanas.

Se creó el lenguaje A, el lenguaje B y se quedaron con el C que era el más potente.

A partir de la 7ª edición de UNIX, AT&T que había adquirido los laboratorios BELL decidió hacerlo comerciable no distribuyendo el código fuente de las siguientes variaciones dándole el nombre de UNIX system V.

A partir de ahí, evolucionó como cualquier otro sistema y ahora pertenece a SANTA CRUZ que es de AT&T.

Como el código fuente era disponible todas las empresas investigadoras tienen su versión con sus mejoras.

CARACTERISTICAS DE LA VERSION 4 (MEJORAS)

- *Conjunto unificado de ordenes (no estándar)*
- *Ofrece un conjunto de Shell´s estándares. Todos ofrecen los mismos : ksh, csh, sh, jsh,rsh (lynux bash)*
- *Sistemas de archivos en red.*
- *Acceso al sistema más sencillo.*
- *Herramientas de administración.*
- *Procesamiento en tiempo real.*
- *Internacionalización del sistema.*
- *Conexiones más sencillas.*

ARCHIVOS Y DIRECTORIOS

El sistema operativo UNIX es un sistema jerárquico, en el encontramos los siguientes elementos :

- ***Archivos :*** *(Estructura básica de almacenamiento de información)*

Desde un punto de vista físico es una secuencia de bytes.

Los nombres de los archivos pueden tener hasta ***256*** *caracteres y pueden incluir una extensión.*

Tienen que ser una secuencia que no incluya el carácter ` / '.

Tampoco es recomendable incluir los siguientes caracteres :

` ! ', ` # ', ` ( `, ` ) ', ` ` ', ` ', ` ; ', `<', `>', Tab,

` $ ', ` ^ ', ` { `, ' } ', ` * ', ` ? ', ` \ `, Espace, Backspace

Solo admiten una extensión.

- ***Directorios :*** *Un directorio es un tipo de archivo que contiene información*

acerca de otros archivos. Se dice que estos otros archivos están dentro del directorio .

La información que contiene es sobre otros archivos. Un directorio puede contener subdirectorios.

El sistema de directorios de UNIX tiene forma de grafo acíclico.

- ***Metacaracteres :*** *El SHELL de UNIX posee un conjunto de operadores*

sobre caracteres que permite especificar más de un valor. Lo más habitual son : ` * '(combinación de caracteres) y ` ?' (cualquier único carácter)

Los metacaracteres son traducidos por el SHELL de forma que la orden no tiene constancia de su existencia.

Ej :

ls (lista el contenido de un directorio)

ls a* ls avion_adios_...etc si avion y adios están en el directorio

Shell ls

El asterisco es mucho más genérico que en MS−DOS :

- **n camión*
- **cion* vacaciones*
- ** nombre.ext*

Tipos de archivos :

Todo lo que trata con información, UNIX lo trata como un archivo.

Para distinguir unas clases de tratamiento de información con otras, UNIX establece los siguientes tipos de archivos :

- ***Archivos ordinarios :*** *Archivo convencional . Colección de palabras de memoria que contienen datos o código.*
- ***Vinculos :*** *Segundo nombre para un archivo ; permite la compartición de información. La existencia de vinculos es lo que convierte al arbos de UNIX un grafo acíclico.*

ARBOL MS−DOS ARBOL ACÍCLICO UNIX

El vínculo es un enlace con el fichero físico, para la compartición de un archivo para varios usuarios.

No se puede crear un archivo vínculo de la nada, es necesario que exista el archivo físico al cual está vinculado.

Ej.

Al realizar un **ls** *el vinculo viene representado como :*

clase − > / etc / passwd

- ***Vínculos SIMBÓLICOS :*** *Amplia el concepto de vinculo permitiendo vincularse a directorios o ficheros desde la red. Un vinculo simbólico desde el punto de vista físico no es más que un archivo que contiene el nombre de otro. No se implementa un vínculo, sino que viene el archivo vinculado. Al hacer un vinculo, crea una entrada más en la tabla de directorios, y al mismo tiempo un archivo al que está vinculado.*
- ***Directorios :*** *Archivo que contiene información sobre otros archivos(es un fichero). Se puede tratar con las ordenes de tratamiento de archivos.*
- ***Archivos especiales :*** *(De dispositivos) Representas dispositivos, hardware de la máquina y se encargan de traducir las ordenes o las llamadas de los comandos UNIX a las secuencias especificas del hardware instalado. Realiza la ocultación de hardware al sistema. Hacen la traducción, funcionan independientemente del hardware en el que está instalado.*

Para saber el tipo de un archivo : **ls −l** *Muestra información larga sobre ficheros .*

Indica al principio del todo con una letra, el tipo de archivo :

d Directorio

l linker (vinculo)

− Archivos ordinarios

s Archivos especiales

Y muestra más información : Tamaño...etc.

PERMISOS Y PROPIETARIOS DE ARCHIVOS :

Es aplicable a los cinco tipos de archivos.

#El borrado de directorios es diferente al borrado de archivos.

En UNIX todos los archivos pertenecen al propietario del proceso que los creó, los archivos del sistema se distribuyen entre varios usuarios del sistema, cada uno de ellos destinado a una labor distinta.

En vez de haber una única cuenta de administrador existen varias cuentas de sistema. Nos podemos conectar como diferentes usuarios que realizan una tarea diferente. Los archivos pertenecen a la cuenta en la han sido creados.

Para cambiar el propietario de un archivo hay que ser el propietario y ejecutar ***CHOWN*** *(chown) el cual regala el archivo. Un no propietario no puede regalar un archivo que no es suyo.*

El cambio de propietario no supone cambio de grupo. El fichero sigue perteneciendo al grupo de su creador. Para cambiar el grupo se utiliza ***CHGRP****(chgrp).*

Los permisos en UNIX son de tres tipos y se establecen a tres niveles :

Tipos : a. r lectura

- *w escritura*
- *x ejecución*

Niveles : a. Propietario

- *Grupo*
- *Resto de usuarios*

El único que puede cambiar los permisos es el dueño y lo hace mediante la orden ***CHMOD****(chmod).*

Existen 9 posibilidades distintas que se representan mediante 9 bits

r w x r w x r w x

Los permisos de un fichero se representan mediante tres números en octal.

Ej. 7 7 7 (Acceso a cualquier usuario)

También es posible indicar los permisos existentes para un fichero en el momento de la creación. Esto se llama máscara de permisos y se establece mediante la orden ***UMASK*** *(umask). UMASK establece la máscara hasta que termine la sesión de conexión.*

Si no hay UMASK 6 6 6 Directorios

7 7 7 Ficheros

Para que la orden UMASK este al iniciar una sesión debemos meterla en el ***.profile*** *(Archivo de configuración del SHELL) KSHELL CASHELL ? ?*

DIRECCIONAMIENTO DE ENTRADA/SALIDA :

El sistema operativo UNIX permite redireccionar tanto la entrada como la salida de cualquier orden sobre ficheros. En realidad un redireccionamiento modifica los valores de los descriptores de archivos sobre los que escriben o leen las ordenes.

Los descriptores de archivos existentes son :

Número	Nombre	Definición
0	Entrada	/dev/tty
1	Salida	/dev/tty
2	Error	/dev/tty

Por defecto están asignados a /dev.

Un redireccionamiento asigna a 0,1,2 un archivo distinto.

Una orden en UNIX lee del descriptor 0, escribe en el descriptor 1, y manda los errores al 2.

Si queremos que escribe lea, o mande los errores a otro sitio se modifica la tabla de descriptores.

El encargado de gestionarlo es el SHELL. El SHELL se encarga de asignar cada número de descriptor al archivo que corresponda.

Los caracteres de redireccionamiento son los siguientes :

> **[fichero]** *Creo fichero y lo lleno con la salida.*

>> **[fichero]** *Añade la salida al archivo.*

< **[fichero]** *Toma la entrada del fichero.*

- **[fichero]** *Error en un fichero nuevo.*

2>> **[fichero]** *Añade la salida de error a un fichero.*

Ejercicio : *Si el comando who me indica quien está conectado y el comando sort ordena un fichero ; ¿Como obtendríamos la lista ordenada de los usuarios conectados ?*

UNIX permite también modificar de una sola vez el descriptor de salida de esa orden y encauzarlo hacia el descriptor de entrada de otra para ello se utiliza el `pipe' ` | '.

Encauzo una orden con otra mediante esta tubería. La tubería crea un archivo temporal que sigue un esquema y sobre el que se garantiza la exclusión mutua.

Ejercicio : *Sacar por pantalla el numero de archivos que contiene un directorio.*

wc −l Cuenta el numero de líneas que escribe.

ls | wc −l Encauzo un listado hacia un contar líneas.

ESTRUCTURA DE LA LÍNEA DE ORDENES :

La sintaxis general de una orden UNIX es :

<comando> [<opciones>] [<fichero_entrada>] [<fichero_salida>]

−letra ó +letra

Muchas ordenes UNIX no admiten ficheros de entrada o/y salida y todos escriben el fichero de entrada o de salida.

Se pueden redireccionar descriptores también con esto.(Todos menos el de errores).

ESTRUCTURA JERÁRQUICA DE FICHEROS :

Como ya hemos dicho, el sistema de directorios de UNIX es de la forma grafo acíclico. Dentro del sistema de ficheros hay dos formas de referenciar un archivo. Llamamos nombre_absoluto al nombre único que se forma siguiendo la estructura de directorios desde la raíz hasta el archivo.

Llamamos nombre_relativo al recorrido desde el directorio actual.

.. Directorio superior

. Directorio actual

*Para cambiar directorios utilizo la orden CD (cd) <directorio> tiene que existir un espacio entre **cd** y el nombre del directorio..*

Para ver el directorio actual : PWD (pwd)

Dentro del tipo de fichero hay un tipo que se llama fichero tubería que es cuando el usuario gestiona el fichero temporal creado para comunicación en una tubería.

MONTAJE DEL SISTEMA DE FICHEROS :

La estructura de directorios de UNIX gestiona los directorios de todos los sistema de ficheros en un único grafo acíclico (Algo del estilo de Windows 95)

El usuario cuando accede a un fichero no tiene constancia de la unidad física del sistema de ficheros físico donde se almacena ese archivo. La ruta de acceso es igual para archivos de todos los sistema de ficheros existentes.

En Windows 95 se conoce aún tratándose como un árbol, dónde se encuentra un fichero.

UNIX posee un sistema de ficheros raíz (root file system) que se corresponde con la unidad principal del sistema. Cuando desea añadir otro sistema de ficheros (administrador) selecciono un directorio dentro de ese sistema de ficheros raíz y sobre él monto mediante MOUNT(mount) el árbol de directorios del nuevo sistema de ficheros. Se superpone la raíz del nuevo árbol sobre el directorio seleccionado.

Se puede seleccionar un sistema de ficheros y montar algo sobre él.

Cuando se monta un sistema de ficheros sobre un directorio, los accesos al directorio se redirigen hacia el nuevo sistema de ficheros con lo que el directorio anterior queda oculto. Por ello es conveniente que el subdirectorio elegido esté vacío.

Para desmontar un sistema de ficheros se usa la orden UNMOUNT(unmount), la cual solo la puede utilizar el administrador.

No sabemos a que fichero estamos accediendo, para saber los ficheros y directorios a los que estamos accediendo utilizamos la orden UNMOUNT(unmount) sin parámetros.

ÁRBOL DE DIRECTORIOS DE UNIX :

/

VAR DEV

USR

ETC HOME SPOOL TMP

BIN LIB SBIN INCLUDE SHARE

MAN

/ var *Contiene archivos que varían entre las diferentes versiones de UNIX y archivos cuyo tamaño cambia en función del instante. Ej. Archivo de correo, archivos temporales de usuario y archivos de registro y contabilidad (Contenidos en /mail, /tmp y /adm respectivamente).*

/ dev *Contiene los archivos especiales o de dispositivos, control de disco, impresoras, terminales...etc.*

/ etc *Contiene archivos de administración y bases de datos de configuración. Son archivos de texto, Ej. passwd, shadow..etc.*

/ home *A partir de este directorio, se encuentran los subárboles de usuario ; los directorios de conexión. En algunos sistemas se llama / users. Nosotros estaremos en / home / alumnos / FM−23 / 951081*

/ spool *Archivos temporales de gestión del sistema, Backup, impresión ...etc.*

/ tmp *Archivos temporales que necesitan las ordenes de UNIX. Los que se crean con tuberías (|)...etc.*

/usr *Contiene los directorios globales que utiliza el usuario en la máquina. Es todo lo que hemos visto hasta ahora /home. (Cualquier editor suele crear un temporal).*

/ usr / bin *Contiene los comandos del sistema y utilidades.*

/usr/sbin *Contiene comandos del sistema destinados a administración ; para el susario son solo de consulta.*

/ usr / lib *Bibliotecas de lenguajes programación.*

/usr/include*Contiene archivos de cabecera del lenguaje de programación C.(.h)*

/usr/share/man *Contiene los ficheros del manual.*

SISTEMA DE FICHEROS DESDE EL PUNTO DE VISTA DEL SISTEMA OPERATIVO UNIX :

El sistema de ficheros interacciona con los procesos de usuario y con el gestor de memoria para realizar su labor.

El diseño de un sistema de ficheros requiere el planteamiento de lassiguientes partes :

- *Intercambio de mensajes entre los procesos implicados.*
- *Organización interna del sistema de ficheros. Cachos de bolques para mejorar las prestaciones y*

descripción del modo de acceso a ficheros.

INTERCAMBIO DE MENSAJES :

Una petición comienza con una solicitud por parte del proceso de usuario (open / close ..etc) cualquier tipo de llamada que accede al sistema.

Llega un petición, el sistema de ficheros se comunica con el núcleo para ver como es la petición, el núcleo con el sistema de ficheros para devolver el resultado, el sistema de ficheros con el gestor de memoria y el gestor de memoria de nuevo al sistema de ficheros... un tratamiento complejo en el que intervienen hasta 29 procesos diferentes.

El sistema de ficheros realiza un bucle de espera de mensajes, y cuando se produce alguna solicitud comienza el tratamiento de las mismas.

ORGANIZACIÓN DEL SISTEMA DE FICHEROS :

El sistema de ficheros está formado por bloques de organización, mapas de bits, nodos−i, directorios y bloques de datos. Estos elementos se organizan según el siguiente esquema :

AUTO ARRANQUE 1 BLK	SUPER BLOQUE 1 BLK	Mapa de bits de nodos−i N BLK´S	NODOS−I N BLK´S	BLOQUES DE DATOS N BLK´S

Autoarranque *El bloque de autoarranque contiene software encargado de cargar el sistema operativo en memoria y empezar a ejecutarlo ; este bloque es al que accede la ROM del ordenador para comenzar a funcionar.*

Super Bloque *Contiene una descripción de la estructura del sistema de ficheros. Sus campos son :*

- *nº de nodos−i.*
- *nº de zonas de datos.* −−>[Author:FCCÖ˜z]
- *nº de bloque del mapa de nodos−i.*
- *Posición de la primera zona de datos.*
- *Máximo tamaño de fichero(nº mágico).*
- *Punteros al bolque del mapa de nodos−i.*
- *nº dispositivo superbloque.*
- *nodo−i del sistema de ficheros montado.*
- *nodo−i del directorio en que se monta.*
- *Última actualización.*
- *Si es de solo lectura.*

Mapa de bits (n−i) *Un nodo−i representa un fichero en UNIX. Existe uno por cada archivo. El mapa de bits de nodos−i, indica si la entrada para ese archivo está libre u ocupada.Cuando creo un archivo se busca un nodo−i cuyo bit esté a 0 y se le asigna ese archivo.*

El borrado de archivos se hace de forma lógica poniendo a 0 el bit del nodo−i del archivo que acabamos de borrar.

Nodos−i *Hay un nodo−i por archivo, y puesto que el número de nodos−i es limitado, también lo es el de archivos. Puedo no llenar el disco, pero quedarme sin nodos−i, y puedo tener mas nodos−i de lo que permite mi disco. Cada nodo−i tiene una estructura de 64 bytes de 16 bits cada uno ; de tal forma que tendremos una estructura de 16x64 bits.*

Los campos contienen :

- *Modo.(Tipo de fichero y bit de protección)*
- *Uid. (Identificador de usuario)*
- *Tamaño.(En bytes)*
- *Última modificación.(nº de dias transcurridos desde 1/1/1970)*
- *Enlaces | GID(Enlaces a directorios que contienen nodo−i)*
- *nº de zona 0*
- *...*
- *nº de zona 9*
- *Puntero indirecto. Punteros a zonas de datos.*
- *Puntero indirecto doble.*
- *Puntero indirecto triple.*

CACHÉ DE BLOQUES :

UNIX mantiene una caché con los últimos bloques accedidos que se organizan en forma de tabla hash y las colisiones se tratan mediante listas doblemente enlazadas.

La salida de bloques de la caché sigue un algoritmo LRU para cada entrada de la tabla hash.

Los bloques de la caché se escriben en disco en dos momentos :

- *Se escriben cuando son expulsador por la CPU.*
- *Cuando se ejecuta la orden SYNC(sync) que copia todos los bloques de la caché a disco. La orden sync se ejecuta aproximadamente cada 5 segundos (o más).*

Existen bloques especiales que se copian a disco directamente.

DECRIPCIÓN DEL MODO DE ACCESO A FICHEROS :

¿Cómo se realiza una búsqueda de un fichero en una ruta ?

Accederemos a los nodos−i de los directorios existentes en la ruta y a los bloques de datos que contienen información sobre cada directorio.

Un directorio en sus bloques de datos contiene una lista de entradas de 16 bytes, de los cuales 2 corresponden al número de nodo−i, y los otros 14 al nombre de fichero correspondiente a ese nodo−i ; cada entrada de la lista se corresponde con un fichero contenido en el directorio.

Ej. Para acceder a /users/practicas.txt

En cualquier acceso se comienza leyendo el nodo−i número 1 que se corresponde con el directorio origen del árbol de directorios. En este nodo−i seleccionamos el puntero a zona de datos que nos indica la zona donde está la información del directorio.

Una zona tiene como mínimo 1024 bytes, si cada entrada es de 16 bytes, nos salen 64 entradas de directorio

en cada bloque o zona. Los directorios que tengan menos de 64 fichero ocuparán un bloque(zona).

Supongamos que 120 es un bloque de datos de directorios :

120

Para acceder a los archivos, accedemos a los nodos−i.

Para saber si una zona de datos es un directorio o un fichero hay que ver el modo en el nodo−i.

INTÉRPRETE DE COMANDOS :

EL SHELL DE PRESENTACIÓN :

Una gran parte del trabajo de UNIX consiste en emitir ordenes. Al emitir una orden, nos estamos comunicando con el SHELL que proporciona gran parte de las características que hacen de UNIX un sistema operativo flexible. El SHELL de UNIX lo podemos ver desde dos puntos de vista diferentes :

- **Intérprete de ordenes :**
    - *Lee la línea de ordenes(desde el prompt hasta el RTM)*
    - *Evalúa los posibles caracteres especiales que contiene, dispone de lo necesario para la ejecución de los procesos de la línea de ordenes ; de tal forma que el SHELL no es el encargado de ejecutarlas.*

- **Lenguaje de programación :**

Ofrece estructuras de alto nivel para crear programas que llamaremos guiones del SHELL (SHELL scripts) que indican una serie de pasos para realizar una labor.

Cuando nos presentamos en el sistema UNIX se inicia automáticamente el SHELL incluido en `/ etc / passwd' para el usuario que se conecta ; este SHELL es el SHELL de presentación, se crea como hijo del proceso ***login*** *y como padre de todos los demás procesos que ejecute el usuario. El SHELL de presentación no termina hasta que el usuario se despide y su terminación provoca el fin de la sesión.*

Nosotros estamos conectados mientras exista el SHELL de presentación.

Una vez que el usuario está conectado, la mayor parte de las interacciones con el sistema toman la forma de dialogo con el SHELL. Este diálogo sigue una secuencia simple y repetitiva :

- *El SHELL presenta un prompt de solicitud de ordenes.*
- *El usuario introduce una orden por teclado.*
- *El SHELL evalúa la orden e indica al núcleo los procesos a crear.*
- *El SHELL espera a que terminen todos los procesos creados y vuelve a 1.*

1 2

4 3

En general la línea de ordenes contiene nombre, argumento y un RTM el SHELL no comienza a procesar la orden hasta que recibe RTM. Si es necesario se pueden realizar varias ordenes con un solo RTM separador por un ` ; '.

TIPOS DE SHELL :

El UNIX sistema V versión 4, proporciona tres tipos de SHELL de presentación, que son :

- *Standar SHELL (Bourne SHELL) / bin / sh*
- *SHELL C / bin / csh*
- *SHELL de Korn / bin / ksh*

El csh y el ksh se desarrollaron para añadir capacidades adicionales al SHELL estándar, entre ellos la edición de la línea de ordenes, los alias de ordenes y el histórico de comandos.

***El SHELL C** fue desarrollado por Bill Joy como una ampliación al sistema UNIX de Berkley. Proporciona todas las características del SHELL estándar y un amplio numero de extensiones. Su sistaxis se ajusta al lenguaje de programación C y es bastante diferente de la del SHELL estándar. Resulta muy cómodo para programar en C. Es incompatible con el SHELL estándar.*

SHELL DE KORN :

El Korn SHELL fue desarrollado por David Korn en 1982 en los laboratorios Bell , incorpora la mayor parte de las ampliaciones del SHELL C y conserva la sintaxis del SHELL estándar. Amplia el SHELL estándar para mantenerlo la misma sintaxis. Los programas de SHELL estándar funcionasen en Korn. Lo del Korn no funcionan en Korn (debido a ampliaciones).

Ficheros de inicialización :

Cuando se inicia el SHELL de presentación, tanto en SHELL estándar como en SHELL de Korn, se busca un archivo llamado .profile en el directorio de conexión. Este archivo es un fichero de texto que funciona como un guión del SHELL y por tanto contiene instrucciones que se ejecutarán en el momento de la conexión. Los guiones del SHELL son ficheros de texto. El SHELL ejecuta ese guión antes de hacer cualquier cosa. Tiene sentencias de información acerca del sistema y de personalización de variables de entorno.

Nuestra terminal va a ser vt220.

El SHELL indica que está listo para recibir la entrada, visualizando el prompt. Por defecto el prompt principal del SHELL es $ (# root).

El SHELL de korn utiliza un archivo adicional al .profile. Este archivo es le que se encuentra en la variable de entorno ENV y se ejecutan después del .profile. Se da el valor a ENV en el .profile. El nombre más habitual es .kshrc ; los fichero que acaban en rc son de configuración. El punto es un archivo de limitación de acceso. El .kshrc es otro fichero de texto.

El fichero de configuración .profile se ejecuta únicamente al iniciar la sesion. El fichero secundario se ejecuta cada vez que se inicializa el SHELL de presentación. Determinados programas tienen la capacidad de generar un escape del sistema con lo que anulan el SHELL de presentación y lo reinicializan al terminar. Ej. editor VI

En esta inicialización es cuando se utiliza el .kshrc pero no el .profile.

Variables del SHELL :

El SHELL dispone de un mecanismo para definir variables que pueden contener información para otros procesos o para si mismo. Ej. TER lo utilizarán muchos procesos. Se pueden definirán conjunto de variables

estándar y también unas variables personales para almacenar cualquier tipo de información.

La variables del SHELL estándar más comunes son :

- *HOME : (Nombre del camino absoluto hasta el directorio de conexión del usuario). Se define automáticamente a partir de ../passwd y tanto el SHELL como otros ficheros pueden hacer usos de ellos (no modificarlo).*
- *PATH : Contiene en orden los directorios en los que el SHELL busca una orden a utilizar , cada nombre de directorio está separado del siguiente mediante dos puntos (:) y el directorio actual se representa con una posición vacía. UNIX busca los archivos en los directorios que vienen en el path en orden (si el directorio actual no se pone en el path, el SHELL no busca en él).*
- *CDPATH : Lista en orden los directorios en los que busca el SHELL para cambiar de directorio con la orden CD(cd) (como un PATH, pero para cambiar de directorio).*
- *PS1, PS2 : Definen respectivamente los prompt principal y secundario ; por defecto PS1 $ y PS2 >*
- *LOGNAME : Contienen el nombre de presentación del usuario.*
- *MAIL : Contiene el nombre del directorio donde se introduce el correo nuevo para el usuario.*
- *SHELL : Contiene el nombre del SHELL de presentación ; se usa para saber que SHELL actúa cuando se produce un escape del sistema.*
- *TERM : Contiene el nombre del terminal empleado y no se fija automáticamente (con los emuladores de terminal actuales a veces se da por defecto).*

Variables más comunes del KSHELL :

- *ENV : Contiene el nombre del fichero secundario de inicialización.*
- *EDITOR : Contiene el nombre del editor de línea de ordenes a emplear.*
- *HISTORY : Contiene el nombre del fichero de histórico de comandos.*
- *HISTSIZE : Contiene el tamaño dedicado al histórico de comandos.*
- *IFS : Contiene el carácter empleado como separador de campos.*

Obtención del valor de una variable.

Para obtener el valor de una variable del SHELL debemos indicar el nombre dela variable y anteponer el símbolo $. Cuando el SHELL encuentra una palabra que comienza por $ supone que es una variable, y sustituye su valor por la parición de la variable.

Ej.

echo $LOGNAME Sustituye $LOGNAME por F951081 echo

ls $HOME sutituye $HOME por /home/alumnos/... ls

Esta forma de acceder a la variable es válida tanto para variables del sistema como para variables creadas por el usuario. Es decir, mediante este tipo de acceso podemos acceder a todos las variables predefinidas como creadas por el usuario.

Para visualizar todas las variables declaradas actualmente en el SHELL ejecutamos la orden SET(set).

Definición de variables en el SHELL :

Aunque existen variables cuyo valor se fija automáticamente hay otras a las que necesitamos asignar un valor. Para definir una variable, empleamos la siguiente sintaxis :

nombre=valor

Se puede cambiar el valor de una variable ya definida.

Si la variable no tiene un valor, la variable no existe. Esta es una orden de asignación y definición al mismo tiempo.

El valor puede contener más de una palabra, en esta caso se pondrá entre comillas. Entre el nombre, el valor y el signo =, no pueden existir espacios en blanco.

Exportación de variables :

Las variables propias o internas al SHELL se distinguen de las que se pueden utilizar en otros programas por el entorno al que pertenecen. En una sesión se crean tantos entornos distintos como procesos se estén ejecutando y cada entorno se divide en entorno propio y entorno global.

Las variables que pertenecen al entorno propio de un proceso solo son accesibles desde ese proceso, mientras que las de entorno global son accesibles desde ese proceso y desde todos los procesos hijos de ese.

Una variable propia del SHELL pasa al entorno cuando es exportada.

Para exportar una variable utilizamos :

export nom_var

Ej.

<Estamos en el SHELL de conexión>

P = 3

echo $P {Sale por pantalla un 3}

ksh {Ejecutamos otro entorno K SHELL}

P = 5

echo $P {Sale por pantalla un 5}

export P {Pasa la variable P al entorno del Ksh de ahora}

ksh {Ejecutamos otro entorno K SHELL}

echo $P {Sale por pantalla un 5}

P = 7

exit {Salgo del 2º Ksh}

echo $P {Sale por pantalla un 5}

{Las modificaciones no afectan al padre}

exit {Salgo del 1er Ksh}

echo $P {Sale por pantalla un 3}

ksh {Ejecuto de nuevo un Ksh}

echo $P {ERROR ! variable no declarada}

Existen variables del sistema que por defecto están exportadas (Pertenecen al entorno global).

Ej : HOME, LOGNAME, PS1, PS2...etc.

Variables noclobber e ignoreeof (en minísculas)

Tanto el Csh como el Ksh utilizan variables especiales llamadas de conmutación para activar o desactivar características. Las variables de conmutación solo pueden tener dos valores : Activado o desactivado. En el SHELL de Korn se gestionan estas variables mediante la orden set con el parámetro −o(activar) y +o(desactivar).

noclobber : *Cuando está activada impide sobreescribir en archivos existentes mediante redireccionamiento de salida. Para confirmar la sobreescritura, utilizamos el carácter |.*

Ej. ls −l > temp {temp existe} No lo hace ERROR.

ls −l >| temp {temp existe} Se fuerza la sobre escritura.

ignoreeof : *La pulsación de CTRL + D equivale a abortar la ejecución. En muchas ocasiones para abortar un proceso hay que pulsar esta combinación varias veces. Esta repetición de teclas puede hacer que además de abortar el proceso, nos salgamos del entorno SHELL de conxión(nos vamos fuera del sitema). La variable ignoreeof cuando está activahace que el SHELL de conexión ignore la pulsación de CTRL + D. Solo protege al SHELL de conexión.*

Histórico de órdenes :

El Ksh mantiene un registro de todas las líneas de ordenes introducidas en una sesion (depende del tamaño que se le de a HISTSIZE). Este resgistro permite desarrollar varias caracteristicas de este SHELL.

Visualización y ejecución :

Para ver en cualquier instante las últimas ordenes ejecutadas, empleamos la orden history ; también permite acceder a una orden en particular dándole como parámetro el código de esa línea de órdenes.

Se puede reejcutar una orden mediante el comando r, como parámetros admite tanto la orden a buscar, como el numero de linea en el que se encuentra la orden.(Sin parámetros ejecuta la última orden).

Ej.

r vi Retrocede hasta encontrar vi, y lo ejecuta.

El nombre de archivo que suele contener el histórico de comandos, es .sh_history

Edición de línea de ordenes :

El ksh, permite elegir un editor de línea de ordenes para modificar y reejecutar ordenes anteriores.

El editor elegido se encuentra en la variable EDITOR y debe ser asignado por el usuario.

Cuando está activo un editor de línea de ordenes podemos remplazar los comandos de ese editor para movernos por ese fichero histórico. El editor más usual es el VI. Otro importante es EMACS.

Cuando está activado el VI como editor disponemos de una ventana de edición de tamaño una línea sobre el fichero histórico.

ALIAS de órdenes :

Un alias de una orden es una palabra que es sustituida por la orden cuando se usa como comando.

Es similar a una variable, pero usada como comando.

Un ALIAS se ejecuta.

Un ALIAS puede servir para dar otro nombre a ordenes existentes, para simplificar la escritura de ordenes o para sustituir a líneas de ordenes muy largas.

Un ALIAS puede hacer referencia solo a un comando, a un comando como parámetro, e incluso a una sucesión de comandos encadenados unos a otros.

Ej.

alias m = mailx

alias lsc = ls −lt

alias cuenta = who | wc−l

Para acceder a un ALIAS, basta con escribir su nombre y pulsar <ENTER>.

La vida de los alias alcanza hasta el final de la sesión.

Para que tengamos un alias desde que se inicia la sesión, hay que meterlo en el .profile.

EJECUCION DE ORDENES EN MODO SUBORDINADO :

Normalmente cuando el SHELL ejecuta una orden permanece bloqueado en espera de que finalice.

Durante este tiempo, no atiende peticiones del usuario.

En ocasiones la ejecución de una línea de ordenes se prolonga durante mucho tiempo y no necesita de la interacción con el usuario. En estos casos resulta útil que el SHELL esté dispuesto para recibir nuevas ordenes antes de que termine la orden anterior.

Esto se consigue haciendo que la ejecución de una orden se realice en modo subordinado :

<background>

Para ello al final de la línea de ordenes añado el símbolo `&'.

No recibe salida de teclado.

Hace que el SHELL no se pare.

La E/S en modo subordinado :

Cuando se ejecuta una orden en modo subordinado, el SHELL la procesa iniciando los procesos necesarios, emite un mensaje de identificación formado por 2 números y a continuación solicita otra orden. Estos dos números representan identificador de trabajo y el PID del proceso. Desconecta la entrada estándar de la orden en modo subordinado del teclado. Una orden en modo subordinado, no puede aceptar ordenes de teclado), pero no desconecta la salida ni el error de la pantalla (cuando genere una salida va a ir a la pantalla). Esto a veces va a resultar molesto, ya que la salida de la orden subordinada se entremezcla con el echo del teclado sobre la pantalla. Resulta muy útil el archivo /d/null , puesto que todo lo que de error se graba en él.

Mantenimiento de trabajos activos.

Lo normal es que una ejecución en modo subordinado sea de operaciones largas.

Una operación muy larga debe continuar, incluso después de que el usuario haya abandonado el sistema.

En condiciones normales cuando un usuario se despide y mantiene trabajos activos, el núcleo elimina este trabajo, con lo cual los trabajos terminan. (Si yo me desconecto el trabajo se elimina)

Para mantener un trabajo activo una vez que la sesión haya terminado se debe ejecutar el trabajo precedido de la orden NOHUP(nohup) esto le indica al núcleo que el trabajo continua incluido después de acabar la sesión, nohup no me obliga a desconectar.

La orden nohup solo se ejecuta en modo subordinado. Todo lo que se ejecute con nohup lo toma como subordinado, la Entrada, la Salida y el Error, se han de direccionar ; si no se hace la Salida y el Error se almacenan en nohup.out.

Órdenes de control de Trabajos.

Para controlar procesos empleamos el PID de los mismos. Cualquier referencia a un PID identifica a un único proceso en el sistema.

Para visualizar los procesos existentes en el sistema utilizamos la orden PS(ps). PS ofrece información sobre que procesos hay, cual es su PID y cuales son sus otras características.

PID nos puede ofrecer todo : El espacio asignado en memoria, el terminal asociado, los padres e hijos del proceso...etc.

En el SHELL estándar, el único control que se puede establecer sobre un proceso, es su eliminación. La orden KILL(kill) envía un número de señal a un proceso. Cuando un proceso envía una señal para la que no está preparado, el proceso termina.

Cualquier tipo de señal puede eliminar un proceso.

Las señales para utilizar con Kill son :

- Interrupción : nº 3
- Terminar : nº 10
- Eliminación incondicional : nº 9

La diferencia está en la fuerza.

1 Cuando hay algún error.

2 Para la finalización.

3 Finalización pase lo que pase Ningún proceso puede controlarlo. Si yo quiero que un proceso acabe siempre, le doy un nº 9.

Ej.

SHELL

señal 3 : La ignora

señal 10 : La ignora

señal 9 : El SHELL termina.

Cuando una señal acaba con el SHELL termina la sesión.

Los procesos que nosotros hagamos podrán capturar cualquier señal menos la 9.

El SHELL de trabajo.

JSH

Es un SHELL destinado para gestionar trabajos de usuario. Nos permite un control más sencillo y más flexible que el SHELL estandar.

Todas las caracteristicas del Jsh están en el ksh.

Dentro del Jsh podemos referenciar a una orden de dos formas :

- *Por el PID.*
- *Mediante el numero de trabajo asignado a esa orden.*

El numero de trabajo es un número único para cada trabajo de un usuario.

Los números de trabajo no van a estar salteados como los PID(Son de un solo usuario).

Un trabajo es un conjunto de ordenes ejecutadas explícitamente por el usuario.

- *Un trabajo no tiene por que ser un proceso. (Puede ser también un conjunto de procesos).*
- *Ejecutamos más procesos que trabajos. (Los trabajos son los procesos o conjunto de procesos que yo he ordenado) .*

El JSH puede :

- *Obtener la lista de trabajos mediante la orden JOB(job).*
- *Pasar un trabajo de modo principal a modo subordinado. BG(bg).*
- *Pasar un trabajo modo subordinado a modo interactivo. FG (fg).*
- *Detener un trabajo. STOP(stop).*
- *Eliminar un trabajo. KILL(kill) Kill −9 <PID> kill %<nºtrabajo>*

LA PROGRAMACION SHELL :

Guiones del SHELL

La palabra SHELL hace referencia a dos cosas, intérprete de comandos, y lenguaje de programación.

El lenguaje SHELL es un lenguaje de programación de alto nivel que permite generar ordenes de UNIX controlando el flujo a través de esas ordenes.

El SHELL solo controla el flujo a través de estas ordenes UNIX.

Los guiones del SHELL se utilizan para hacer referencia a un conjunto de ordenes que se ejecuten de la misma forma múltiples veces. Pero todo lo que va dentro de un guión se puede escribir directamente en la línea de ordenes.

Un guión es un fichero de texto.

Ejecución de un guión

Para hacer ejecutable un guión es necesario conceder autorización de ejecución sobre el archivo.(No hay que compilar).

Haciendo ejecutable un fichero puedo usarlo como una orden de la línea de ordenes. Este modo de ejecución provoca que el SHELL cree un subshell destinado a leer y ejecutar el contenido del guión.

Para hacer una ejecución se crea un SUBSHELL igual que el padre. Este SUBSHELL coge las líneas de una en una al fichero ejecutable.

Todo lo que hay en un guión está en la línea de ordenes. Esto mismo se puede conseguir ejecutando un proceso SHELL que precede como parámetro el nombre de un guión.

Ej

PR 1 ! Ha de tener permiso de ejecución.

Ksh PR1 ! No necesita permiso de ejecución.

A veces resulta útil ejecutar un conjunto de ordenes en un SUBSHELL sin necesidad de crear un guión.

Para ello debo poner la lista de ordenes separadas por [;] y entre paréntesis. Cualquier redireccionamiento para las listas entre paréntesis afecta a todas las ordenes de la lista.

Cuando ejecutamos guiones en un SUBSHELL, todas las variaciones del entorno que generen, afectan al SUBSHELL.

Cuando deseemos que un guión del SHELL cambie el entorno del SHELL en el que dicho guión se ejecuta y

no del SUBSHELL, podemos ejecutarlo de las dos siguientes formas :

- *Agrupando las ordenes entre llaves (Afecta el SHELL actual)*
- *Mediante la orden . (dot) seguida de un nombre de guión el guión se ejecuta en el SHELL principal.*

Si ejecutamos el .profile : . .profile.

Comentarios en los guiones del SHELL.

Para incluir un comentario en un guión del SHELL debo incluir el carácter # (almohadilla). El SHELL considera comentario todo lo que aparezca detrás de la almohadilla hasta el salto de línea. No puedo insertar comentarios dentro de una línea, sino que tengo que ponerlo al final de la línea.

Parámetros posicionales .

Los guiones del SHELL son capaces de recibir parámetros en la llamada, para que el guión haga algo diferente.

Para usar parámetros, el SHELL utiliza unas pseudovariables llamadas parámetros posicionales cuyo valor se fija automáticamente al ejecutarse el guión.

Los más importantes son :

$# : Indica el numero de argumentos.

$1,$2,$n.. : Especifica 1º, 2º, enésimo argumento.

$0 : Especifica el nombre del programa.

$: Da la lista completa de argumentos sin el $0.*

Para reordenar los parámetros posicionales utilizo la orden SHIFT(shift). Desplazar hacia la izquierda los parámetros eliminando el primero y disminuyendo el número en uno.

Para asignar un valor a los parámetros posicionales de un guión utilizo la orden SET(set) seguida de una ejecución. Esto hace que la salida de la ejecución se asigne sobre los parámetros posicionales del guión.

En la secuencia de cadenas de ejecución de un guión interviene $? Que contiene el valor devuelto por la última sentencia de guión ejecutable.

Otras variables del Guión.

Un guión posee por defecto todas las variables externas del SHELL que lo ejecutó. Además podemos definir cualquier otra variable de la misma forma que el SHELL.

Esa variable desaparece cuando desaparece el guión.

Dentro de un guión podemos aplicar dos operadores adicionales para asignar valor a la variable en case de que esta no lo tenga.

Si lo que hacemos es una consulta la variable no declarada toma este valor por defecto.

Existen dos formas :

- *$ {nom_variable :− valor por defecto}*

Si la variable no tiene valor, devuelve el valor por defecto pero no se lo asigna}

- *${nom_variable := valor por defecto}*

Igual que :− pero si que se lo asigna.

Ej.

$0 = pr1

P=3

echo ${P :=7} devuelve 7

echo ${D :=5} devuelve 5

echo $D devuelve 5

echo ${E :−3} devuelve 3

echo $E devuelve no definido

SENTENCIAS DE CONTROL EN LA PROGRAMACIÓN SHELL

Dentro de la programación del SHELL voy a utilizar ordenes de la línea de comandos.

Operaciones lógicas :

Evaluación de expresiones lógicas : *Se realizan mediante la orden test (TEST). Test permite realizar comprobaciones sobre enteros, cadenas de caracteres y estado de ficheros. Si la comparación es cierta, test genera un código de retorno 0 y si es falsa, distinto de 0.*

*Los **test** admitidos son :*

- *Sobre enteros :*

n1 −eq n2 =

n1 −en n2 "

n1 −gt n2 >

n1 −ge n2 >=

n1 −lt n2 <

n1 −le n2 <=

- *Sobre cadenas :*

−z cad Si la longitud de la cadena es = 0.

−n cad Si existe la cadena.

cad1 = cad2 cad1 igual a cad2

cad1 != cad2 cad1 distinto de cad2.

Cadena Si la cadena es distinto de la cadena vacía.

- *Sobre ficheros* :

−a fichero Si existe el fichero.

−r fichero

−w fichero Permisos del fichero.

−x fichero

−f fichero Fichero ordinario.

−d fichero Directorio.

−h fichero Vínculo.

−c fichero Especial de carácter.

−b fichero Especial de bloque.

−p fichero Tubería.

−s fichero Tamaño > 0.

La orden test permite también concatenar expresiones lógicas mediante los operadores :

−a : AND

−o : OR

! : NOT

Ej.

test $# −gt 5 −a −w salida

Si el número de parámetros del guión es mayor que cincoy se puede escribir en el fichero salida

Una sintaxis alternativa para test consiste en situar la expresión lógica entre corchetes.

El SHELL de Korn, proporciona la orden [[ ]] que amplia el funcionamiento de test. Una expresión lógica entre doble corchete permite utilizar operadores clásicos y trata correctamente las variables nulas, es decir si una variable no es nula lo compara con la variable vacía.

ÓRDENES DE CONTROL

IF ...THEN

Sintaxis :

if orden **if** orden

then órdenes **then** órdenes

fi else órdenes

fi

El if ejecuta la orden de antes del then y si el resultado devualto es 0, ejecuta el then. Si el resultados es distinto de 0 o bien ejecuta el else, o nada.

CASE

Sintaxis :

case cadena

in

(patron/lista) orden

orden

; ;

(patron/lista) orden

orden

; ;

esac

Compara la cadena con cada caso, y si alguno coincide, ejecuta las ordenes asociadas. Cuando ejecuta las ordenes, sale del case.

(patron/lista) es un valor, o una lista de valores.

*Cualquier cadena con un asterisco(*) sería el else. Si se pone el *, se debe poner siempre al final.*

SELECT

La sentencia select permite comparar una cadena con un conjunto de casos, pero obliga a que la cadena se corresponda con algún caso.

Por tanto lee un valor de la entrada estándar y si no hay concordancia continua solicitando hasta que lo haya.

FOR

Sintaxis :

for variable

in lista

do

órdenes

done

Realiza una iteración para cada elemento de la lista, asignándoselo como valor a la variable.

Si no hay una lista, itera sobre los parámetros posicionales.

WHILE

Sintaxis :

while orden

do

órdenes

done

UNTIL

Sintaxis :

until orden

do

órdenes

done

Las ordenes de interrupción de bucle : Si se desea interrumpir un bucle independientemente de la condición el SHELL ofrece dos órdenes :

- ***Break :*** *Finaliza la interación y continúa con la siguiente instrucción.*

- ***Continue :*** *Finaliza la iteración y vuelve a la condición o comparación.*

Las dos permiten un parámetro

Órdenes true y false :

Son 2 órdenes constantes que devuelven un valor 0 y distinto de 0 respectivamente.

OPERACIONES ARITMÉTICAS

El SHELL estándar permite realizar operaciones aritméticas mediante la orden ***expr****.*

La orden ***expr*** *recibe tres parámetros (dos operadores y un operador) y muestra en la salida estándar el resultado.*

Para realizar operaciones más complejas se hace uso del operador ***grave*** *también llamado restitución de ordenes y que se representa con dos comillas invertidas. El operador grave ejecuta el comando entre comillas y sitúa la salida del mismo en lugar del comando. Permite que la salida de una orden forme parte de la línea de órdenes.*

Ej.

expr `expr 1+2`+1

El Kshell facilita las operaciones aritméticas mediante la orden let (LET). Let permite operar con variables directamente, sin poner $, utilizan más de un operador por línea, realizan operaciones más complejas y realizan comparaciones entre enteros. Una alternativa del let es (( ))

Ej.

let x = 2*y%z " x = `expr ` expr 2*y `opr $z`

Otras órdenes del SHELL :

Órdenes de E/S :

read *Lee una línea de la entrada estándar y se la asigna a una o más variables de tal forma que la primera palabra se asigna a la primera variable, la segunda a la segunda,. Y el resto de línea a la ultima variable.*

El SHELL de korn permite combinar una petición de entrada con la lectura de variable.

Ej.

read A B ? Entrada :

print *Sustituye en el SHELL de korn el echo y permite ampliar los formatos de este.*

<u>Otras órdenes :</u>

trap *La orden trap permite especificar una secuencia de acciones a realizar cuando se recibe una señal :*

Sintaxis : trap <conjunto de ordenes> <números de señal>

Ej. trap rm tmp$$ 2 3 15

Cuando llegue la señal 2, 3, 15 se ejecutarán las órdenes.

La señal 9 no se puede capturar.

Exit *Provoca la finalización de un proceso puede recibir un parámetro entero que se corresponde con el código de retorno que generará el proceso al terminar. Si no hay parámetro, se devuelve un cero. Los códigos de retorno que se devuelven por convenio son :*

0 terminación correcta.

1 terminación anormal.

2 error en los parámetros.

Xargs *Muchas ordenes de UNIX no pueden recibir sus parámetros desde la entrada estándar lo que impide que otra orden pueda pasárselos mediante una tubería. La orden xargs permite redireccionar la salida de una orden como parámetros de otra.*

Su sintaxis es :

xargs [*indicadores] [orden[(argumentos iniciales)]]*

Admite dos indicadores :

−i toma cada línea de la entrada estándar y realiza una ejecución de la orden para esa entrada.

−p pide confirmación.

MATRICES Y FORMA DE ACCESO

El ksh permite agrupar variables para formar un vector.

Solo admite matrices bidimensionales.

Los valores que forman una matriz tienen en común la referencia a un único nombre base de matriz entre los valores de la misma.

Los elementos de una matriz no tiene por que ser del mismo tipo.

En UNIX el concepto de variable no tiene declarado un tipo.

Las matrices no son más que un elemento lógico del programador.

Para asignar valores a un elemento de una matriz se utiliza la siguiente sintaxis :

PR[PAT] = 30

PR[ZAN] = 15

PR[LECH] = `no se'

Para acceder al valor de una matriz se hace mediante la siguiente sintaxis :

echo ${PR[PAT]}

Las matrices solo tiene interés cuando relaciones elementos.

Ej :

for I

in `cat hortalizas'

do

echo ${PR[$I]}

done

LA HERRAMIENTA AWK :

Es una de las cosas más fáciles de explicar y con lo que más problemas vamos a tener.

Es una herramienta de programación.

Se aproxima a la programación funcional.

Se denomina programación por patrones.

Se divide en patrones y acciones. (Acciones asociadas a patrones)

Acciones Lenguaje C

Awk se encarga de leer la entrada (normalmente la estándar). La divide en registros. (Cada registro es una linea Separador de registros(!))

Para cada registro buscamos concordancia con algún patrón, si se produce concordancia ejecuta la acción asociada al patrón sobre el registro.

Awk es un traductor. (La entrada puede ser una cosa, y la salida otra)

La sintaxis de awk es de dos tipos :

awk patrón {acción} ; patrón {acción} ;...´ [fichero de entrada]

awk −f fich.prog [fichero de entrada]

−f Indica separador de campo.

−v Asignación de valor.

PATRONES :

Los patrones son los encargados de resolver la entrada.

Funciona como un gigantesco CASE.

Si no hay patrón para una acción se ejecuta sobre todas las líneas de ficheros.

Patrón vacío es cualquier o ningún registro.

Tipos de patrones :

- Patrón constante : *Son cadenas de caracteres fijas situadas entre barras ` / ' y para las que se busca la aparición en cualquier punto del registro.*

Ej. /patata/ {print} Busca la cadena patata en cualquier posición

- Expresiones regulares : *Son combinaciones de caracteres y operadores de carácter. Los operadores permitidos son*

^ Principio de línea.

$ Fin de línea.

[ ] Clase de caracteres.

| OR

* Cero o más apariciones.

+ Una o más apariciones.

? Cero o una aparición.

. Comodín.(Un carácter)

( ) Agrupación.

− Rango.(Utilizando caracteres ASCII)

Ej. Localice en el fichero passwd aquellos usuarios cuyo número de expediente es impar y pertenecen al grupo 109.

Awk `^f.....[13579] :.* :109 {print} [etc/passwd]

- Comparación de cadenas : *Permiten ejecutar acciones en función de determinados valores del registro de entrada. Para acceder a partes del registro se definen las variables$1, $2...$199 que contienen automáticamente el primero, segundo...etc campo del registro. Los comparadores admitidos son :*

~ Identificación : Comprueba si una cadena se ajusta a un patrón.

!~ No identificación.

== Igualdad.

!= Desigualdad.

<,>... Comparación.

- Patrones compuestos : *Se obtienen combinando patrones simples mediante los operadores :*

&& AND.

|| OR

! NOT.

- Patrones de rango : *Se forman con dos patrones separados por una coma. Awk ejecuta la acción sobre todos los registros de la entrada situada entre el registro que coincide con el primer patrón y el que coincida con el segundo.*
- Patrones BEGIN y END : *Son dos patrones especiales de awk. La acción asociada al patrón BEGIN se ejecuta antes de leer ninguna línea de la entrada. Se utiliza para inicializaciones. La acción asociada a END se ejecuta después de leer el fin de fichero ; se utiliza para presentaciones de resultados. Ninguno de los dos utiliza un registro en una línea para la acción. (No se puede hacer referencia a $1, $2 ..en BEGIN o END porque no hay registros contenidos)*

ACCIONES :

Las acciones en awk son operaciones en lenguaje C que utilizan los campos del registro que concuerdan con el patrón.

La acción vacía es equivalente a un print.

Variables :

Awk utiliza la misma nomenclatura de variables que C, pero no exige que una variable esté declarada para poder usarla.

Para poder operar con una variable debe tener un valor.

Además de las variable definidas por el usuario, awk puede acceder a todas las variables del SHELL situándolas entre comillas.

Awk también ofrece un conjunto de variables predefinido. Las más importantes son :

***FS** Separador de campo.*

***NF** Número de campos del registro.*

***NR** Número de registros leídos.*

***FILENAME** Contiene el fichero de entrada.*

***ARGV** Array de argumentos de la llamada a awk.*

Asignación :

−v nombe = valor

Operadores :

Permite operadores sobre caracteres y sobre enteros.

Los operadores permitidos son entre otros :

- ***Aritméticos :*** *+, −, *, /, %,^...etc.*
- ***De asignación :*** *=, +=, −=, ^=, %=...etc.*
- ***Operadores de comparación :*** *==, >, <, >=, <=, !=*
- ***Funciones :*** *tan, sen, cos, log, exp, sqrt, rand...etc.*
- ***Funciones sobre cadena :*** *substr, match, length, split...etc.*
- ***Operadores lógicos :*** *&&, ||, !*
- ***Unión de dos cadenas :*** *Se pone una detrás de la otra.*

Arrays :

La definición de matrices en awk es idéntica a su definición en UNIX ; Un conjunto de valores que no tienen relación de tipo se encuentran unidos lógicamente por un elemento base y un conjunto de índices.

Los arrays son unidimensionales. Los índices pueden ser cualquiera.

Para acceder a un elemento de array tanto en asignación como en obtención de valor se hace uso de la sintaxis de C, que es la misma que la de pascal.

Para simplificar la gestión de arrays awk ofrece las siguientes estructuras :

- *delete <BASE> [<INDICE>] Elimina un elemento del array.*
- *<subíndice> in <BASE> Es cierto si el subíndice existe.*
- *for <VARIABLE> in <BASE> : Realiza una iteración por índice.*

sentencia

Funciones definidas por el usuario :

La definición de una función utiliza la sintaxis de C, pero sin tipo.

Sintaxis :

function <nombre> (lista de parámetros)

{lista de sentencias}

Aquí no hay tipos, pero puede devolver un valor con la sentencia return. (Si incluye return es función , sino es procedimiento).

Sentencias de control :

Las sentencias de control de flujo son :

- **if** (condicion) sentencia [**else**][sentencia]

- **while** (condición) sentencia
- **do** (sentencia) **while** (condición)
- **for** (inicialización ; test ; incremento) sentencia
- **break** Fuerza la salida del bucle.
- **exit** finalización de la entrada.

Funciones de UNIX TEMA 5

La sintaxis de las funciones UNIX es :

function nombre

{

orden ;

orden ; Ordenes UNIX (Variables con $)

...

orden ;

}

Donde las ordenes se correspondes con comandos del SHELL. Una vez que se ha definido una función en un SHELL esta es accesible solo desde ese SHELL. Los parámetros de esa función se referencian desde dentro de la misma mediante las pseudo variables $1, $2...etc.

Los parámetros son iguales que en los guiones.

No existe la exportación de funciones.

Las funciones desaparecen cuando desaparece el SHELL.

E/S en awk :

Entrada :

Awk recorre automáticamente la entrada estándar analizando cada registro y comparándolo con los patrones.

*En ocasiones es necesario leer de forma automáticamente alguna línea de la entrada estándar o de otro fichero. La función **getline** toma una línea de la entrada estándar y almacena su valor en una variable.*

Esta lectura es aparte de la que realiza el awk .

Lee de la entrada estándar una línea y la almacena en una variable.

Salida :

*Para generar salida awk dispone de las funciones **print** y **printf** las dos escriben en la salida estándar pero **printf** permite dar formato a la salida.*

Tanto la entrada como la salida puede redireccionarse a un fichero distinto del estándar mediante >,>> y < con el fichero entre comillas.

El fichero de terminal se especifica como /dev/tty

Ejemplos :

- Dada una tabla como la siguiente :

	ENERO	FEBRERO	...	DICIEMBRE
PERAS	10	20	...	7
UVAS	15	7	...	40
...	...	...	...	...
MANZANAS	15	20	...	30

Generar mediante awk un fichero de salida que muestre la misma tabla y calcule el total anual de cada fruta, el total de ventas por cada mes y el total general.

Solución :

Necesitamos tres patrones diferentes :

1º. Patrón para la cabecera: [^0−9] ó ^[\ t] ó NR ==1

2º. Patrón para el resto: [^0−9] ó NR !=0

3º. Patrón para la última línea: end

Los cuales tiene sus respectivas acciones :

1º { **print** $0 \t TOTAL; // sino se pone acción asume un print.

for (i=2 ; i < NF ; i++) // Para inicializar la matriz.

total [i] = 0 ;

}

2º { suma = 0 ;

for (i=2 ; i < NF ; i++)

{

suma = suma + $i

total[i] = total[i] + $i

}

print $0 \ t suma ;

}

3º { suma = 0 ;

for (i in total)

{

linea = linea \ t total [i] ;

suma = suma + total [i] ;

}

print linea \ t suma ;

}

NOTA : Es bueno poner ; después de todas las acciones.

- Realizar mediante awk un corrector ortográfico que elimine palabras duplicadas y consecutivas. Para cada palabra duplicada solicitará confirmación interactivamente.

<patrón_vacío>

// acción

{

if (ant == $1)

{

print Palabra $1 duplicada, ¿Eliminar ? > /dev/tty

getline resp < /dev/tty ;

if resp = N **printf** (%s \ n, ant)

else printf(\ n)

}

for (i=1 ; i <NF ; i++)

if ($i == $(i+1))

{

print Palabra $1 duplicada, ¿Eliminar ? > /dev/tty

getline resp < /dev/tty ;

if resp = N **printf** (%s, $i) ;

else printf(%s, $i) ;

}

ant = $NF ;

}

BEGIN {

ant =

}

END {

printf (%s, $i)

}

TEMA 7: GESTIÓN DE PROCESOS.

Control de procesos:

Un proceso en UNIX es una instancia de programa en ejecución.

Cuando ejecutamos un programa, creamos uno o varios procesos.

Un proceso son varias estructuras de datos.

Desde un punto de vista más a bajo nivel un proceso es el resultado de la ejecución de una llamada al sistema; La llamada al sistema ***fork.*** *Cuando un proceso ejecuta la llamada fork, el sistema operativo crea una copia de la memoria virtual del proceso llamador. Se crea un clónico del primero.*

La diferencia entre las dos copias es el PID de cada proceso. Por esto es un proceso distinto.

En UNIX se establece una metáfora al considerar los procesos como seres vivos. Un proceso nace, muere, tiene hijos ..etc.

Prioridades de procesos:

El núcleo de UNIX reparte entre todos los procesos recursos, según sea el sistema de prioridades. Este es el que manda en los recursos. La prioridad de un recurso la determina el administrador en función del propietario del mismo y el usuario normal solo puede influir ligeramente a este valor de prioridad. La prioridad de un proceso sigue una formula así:

PRIORIDAD = PRIORIDAD BASE + VALOR NICE

La prioridad base es la que viene marcada por el tipo de usuario. El valor nice es el valor modificador que puede emplear el usuario.

El usuario puede asignar valores nice desde −1 hasta −19 mediante la orden nice seguida del valor.

La orden nice solo permite disminuir la prioridad de un proceso con lo que aumenta el tiempo de CPU destinado a los demás procesos.

Si se pudiese aumentar la prioridad, sería un caos.

El administrador puede utilizar valores nice positivos poniendo dos signos menos seguidos.

Ej:

nice pr1 # −10 Usuario y administrador.

nice pr1 # − − 10 Solo administrador.

Ordenes de tratamiento de procesos:

Cuando nosotros le damos una orden al SHELL, este se encarga de ejecutarla.

Un proceso puede esperar la llegada de una señal de terminación de hijos en ejecución mediante la llamada al sistema wait. UNIX ofrece un comando wait que permite a un guión del SHELL realizar una espera equivalente a la llamada al sistema.

Para mantener información acerca de los procesos hacemos uso de la orden PS. Trabaja con el PID de procesos. El sistema operativo asigna los PID de proceso de forma consecutiva e incremental y cuando llegue al último número válido reutiliza los números de procesos que han dejado de existir. El único proceso que no carga es el 0.

*Un modo sencillo de afectar a la planificación consiste en dormir un proceso durante un intervalo de tiempo. La orden **sleep** recibe un parámetro, y luego duerme el proceso que lo ejecuta durante ese tiempo.Lo que hace es mandar una llamada al sistema para que se duerman. Para un proceso significa ayudar a los demás.*

Sleep permite dormir un proceso pero pierde precisión en intervalos de tiempo largo.

En UNIX por defecto un bloque son dos sectores y un sector son 512 bytes. La memoria del disco se asigna no por bloques sino por zonas, donde una zona el 2n bloques, tal que n es un valor que decide el administrador cuando crea el sistema de ficheros.

Apéndice CSHELL al final (Independiente del KSHELL)

VINCULO

VINCULO

Archivo Físico

PROPIETARIO

GRUPO

RESTO

(120)

(6)

(132)

1 .

6 .

1 ..

1 ..

4 bin

...

7 dev

26 prácticas.txt

14 etc

132

6 user

8 tmp

9 spool

...