Fundamentos de Linux

Código de Curso: CY310

Versión: 3

Shell
(Intérprete de Comandos)
y
Procesos en Linux

Fundamentos del Shell

Objetivos de Aprendizaje

- Proporcionar una visión general del shell de Linux
- Discutir acerca de las variables shell
- Describir los metacaracteres
- Explicar cómo asignar y referenciar las variables shell

Introducción al Shell de Linux

- El Shell ejecuta un conjunto de comandos Linux y procesa las entradas del usuario
- Proporciona una interfaz al sistema operativo subyacente
- El kernel maneja las interacciones complejas con el hardware del sistema
- El shell actúa como un intermediario entre el usuario y el kernel

Responsabilidades de un Shell

- Controlar y asignar trabajos
- Manejar más de un proceso a la vez
- Redireccionar la entrada y salida estándar
- Mantener un historial de comandos
- Proporcionar un lenguaje de comandos para escribir *shell scripts*

Variables Shell

- Las variables shell mantienen el registro de diferentes tipos de información
- Son locales al shell en el que están definidas
- Cada shell tiene su propio conjunto de variables que pueden ser fijadas y referenciadas por el usuario

Variable PATH

- La variable **PATH** en Linux se establece en todos los directorios del sistema donde los comandos estén disponibles para ejecución
- La información disponible en **PATH** consiste de la ruta completa de los directorios donde los comandos están ubicados
- **PATH** es sólo una de las muchas variables predefinidas que mantiene un shell

IBM

Variables Shell Definidas por el Usuario

- El shell permite a los usuarios crear sus propias variables
- Para crear una variable, el usuario debe ingresar

```
name=value
```

  - ✓ **Name:** es la variable shell definida por el usuario
  - ✓ **Value:** es el valor asignado a la variable shell definida por el usuario

Metacaracteres

- Los metacaracteres son caracteres que tienen un significado especial para el shell
- Los metacaracteres se agrupan en las siguientes categorías:
  - ✓ Sustitución de Comodines (Wildcard)
  - ✓ Redirección y tuberías (pipes)
  - ✓ Notación de línea de comandos
  - ✓ Sustitución de parámetros
  - ✓ Escapes y comillas

Sustitución de Comodines

- Los caracteres de comodín son un subconjunto de metacaracteres que se usan para buscar y relacionar patrones de archivo
- Los cuatro caracteres comodín son:

`?`	Corresponde a cualquier carácter en un nombre de archivo
`*`	Corresponde a cualquier cadena de cero o más caracteres en un nombre de archivo
`[list]`	Corresponde a cualquier carácter de **`list`**
`[^list]`	Corresponde a cualquier carácter que no está en **`list`**

Uso de ?

```
[mickeymouse@mycomputer mickeymouse]$ ls -o

total 34
-rw-r--r--   1 mickeymo    13 Jul   3 18:20 lint.txt
-rw-r--r--   1 mickeymo     7 Jul   3 17:53 linux.txt
-rw-r--r--   1 mickeymo    12 Jul   3 17:40 linux1.txt
-rw-r--r--   1 mickeymo    13 Jul   3 17:40 linux2.doc
-rw-r--r--   1 mickeymo    16 Jul   3 17:39 linuxChap1.doc
...
```

Uso de ?… 2

```
-rw-r--r--   1 mickeymo    17 Jul   3 17:39 linuxChap2.doc
-rw-r--r--   1 mickeymo     9 Jul   3 17:39 linuxCont.doc
drwxr-xr-x   2 mickeymo  4096 Jul   3 17:40 linuxInternals
-rw-r--r--   1 mickeymo     7 Jul   3 18:53 listing.lst
-rw-r--r--   1 mickeymo    37 Jul   3 18:05 mist.doc
-rw-r--r--   1 mickeymo    12 Jul   3 18:59 nest.lst
drwxr-xr-x   2 mickeymo  4096 Jul   3 17:40 unixInternals

[mickeymouse@mycomputer mickeymouse]$
```

ls -o muestra un listado largo de archivos sin agrupar la información.

```
[mickeymouse@mycomputer mickeymouse]$ ls -o linux?.txt
-rw-r--r--    1 mickeymo    12 Jul   3 17:40 linux1.txt
[mickeymouse@mycomputer mickeymouse]$
```

IBM

Uso de *

```
[mickeymouse@mycomputer mickeymouse]$ ls -o linux*
-rw-r--r-- 1 mickeymo  7 Jul 3 17:53 linux.txt
-rw-r--r-- 1 mickeymo 12 Jul 3 17:40  linux1.txt
-rw-r--r-- 1 mickeymo 13 Jul 3 17:40  linux2.doc
-rw-r--r-- 1 mickeymo 16 Jul 3 17:39  linuxChap1.doc
-rw-r--r-- 1 mickeymo 17 Jul 3 17:39  linuxChap2.doc
-rw-r--r-- 1 mickeymo  9 Jul 3 17:39  linuxCont.doc
linuxInternals:total 0
[mickeymouse@mycomputer mickeymouse]$
```

IBM

Uso de [list] y [^list]

```
[mickeymouse@mycomputer mickeymouse]$ ls -o li[ns]t*
-rw-r--r--1 mickeymo 13 Jul  3 18:20 lint.txt
-rw-r--r--1 mickeymo  7 Jul  3 18:53 listing.lst
 [mickeymouse@mycomputer mickeymouse]$
```

Busca **n** ó **s** en la tercera posición

```
[mickeymouse@mycomputer mickeymouse]$ ls-o *[12C]*
-rw-r--r-- 1 mickeymo 12 Jul 3 17:40 linux1.txt
-rw-r--r-- 1 mickeymo 13 Jul 3 17:40 linux2.doc
-rw-r--r-- 1 mickeymo 16 Jul 3 17:39 linuxChap1.doc
-rw-r--r-- 1 mickeymo 17 Jul 3 17:39 linuxChap2.doc
-rw-r--r-- 1 mickeymo  9 Jul 3 17:39 linuxCont.doc
```

Descriptores de Archivos

- El shell asigna tres descriptores estándar a cada programa al iniciarse:
  - ✓ **STDIN**
  - ✓ **STDOUT**
  - ✓ **STDERR**
- Los descriptores se asignan con un valor como sigue:

```
STDIN    0     StandardInput   (Teclado)
STDOUT   1     StandardOutput  (Monitor)
STDERR   2     StandardError   (por defecto se
                               asigna al monitor)
```

Redirección y Tuberías

Dos otros usos de la redirección son **`n>`** y **`>&n`**.

Uso de `n>`

- **`'n'`** representa el descriptor de archivo
- El nombre de archivo a continuación del > es el archivo destino al que se enviará el resultado.
- Los mensajes de error son redireccionados a la salida, por ejemplo el monitor

Uso de n>… 2

Si se quiere redireccionar el error a un archivo se puede usar el operador > como sigue:

```
[mickeymouse@mycomputer mickeymouse]$ cat linux.txt file1.txt
Linux is an operating system.
It follows the GNU standard.
Linux provides users with a command language.
The kernel of Linux is the core of the operating system.
Linux is a powerful and versatile operating system.
cat: file1.txt: No such file or directory
[mickeymouse@mycomputer mickeymouse]$
```

Debido a que no se tiene un *file1.txt* se obtiene el mensaje de error correspondiente, después que se muestran las líneas de *linux.txt*

Uso de n>… 3

Si se quiere capturar el mensaje de error en otro archivo, se hace lo siguiente :

```
[mickeymouse@mycomputer mickeymouse]$ cat linux.txt
file1.txt 2>error
Linux is an operating system.
It follows the GNU standard.
Linux provides users with a command language.
The kernel of Linux is the core of the operating system.
Linux is a powerful and versatile   operating system.
[mickeymouse@mycomputer mickeymouse]$
```

Uso de >&n

>&n se usa cuando tanto la salida como el error estándar deben redireccionarse al mismo archivo.

El shell proporciona el mecanismo para combinar estos dos flujos.

```
[mickeymouse@mycomputer mickeymouse]$
cat linux.txt file1.txt > outfile 2>&1
[mickeymouse@mycomputer mickeymouse]$
```

No se verá ninguna salida en la pantalla. Tanto la salida estándar como el error estándar han sido redireccionado al archivo outfile.

Notación de Línea de Comandos

- Cierto número de metacaracteres se pueden usar en una línea de comando para realizar tareas especiales en el shell.
- Estas son:
  - Se pueden ingresar comandos en múltiples líneas usando `\`
  - Se pueden ejecutar los comandos en segundo plano usando `&`
  - Se puede ingresar más de un comando en una sola línea usando `;` como separador
  - Los comandos se pueden agrupar usando `()`
  - Se pueden aplicar operaciones condicionales en los comandos, usando `&&` y `||`

Uso de \

El metacaracter ****, seguido por la tecla **<Enter>**, permite al usuario ingresar los parámetros del comando en múltiples líneas

- Cuando se presiona **** y la tecla **<Enter>**, un símbolo **>** aparece en la siguiente línea para indicar que el comando no está completo
- Continuar el comando en la siguiente línea es útil cuando los argumentos del comando son largos

```
[mickeymouse@mycomputer mickeymouse]$ cat > \
> myfile.txt
      This is a new file.
      ctrl-d
[mickeymouse@mycomputer mickeymouse]$
```

El metacaracter **** le dice al shell que espere por una entrada adicional

Uso de &

El metacaracter **&** se usa para ejecutar el comando precedente en segundo plano

```
[mickeymouse@mycomputer mickeymouse]$ monitor & \
> cat linux.txt
[1] 2259
Linux is an operating system.
It follows the GNU standard.
Linux provides users with a command language.
The kernel of Linux is the core of the operating system.
Linux is a powerful and versatile operating system.
Number of users in the system: 204
[1] Done monitor
[mickeymouse@mycomputer mickeymouse]$
```

Uso de ;

El metacaracter ; se usa para separar comandos cuando se ingresa más de un comando en la misma línea

```
[mickeymouse@mycomputer mickeymouse]$ ls -o; cat
linux.txt
total 34
-rw-r--r--   1 mickeymo   13 Jul  3 18:20 lint.txt
-rw-r--r--   1 mickeymo    7 Jul  3 17:53 linux.txt
-rw-r--r--   1 mickeymo   12 Jul  3 17:40 linux1.txt
-rw-r--r--   1 mickeymo   13 Jul  3 17:40 linux2.doc
-rw-r--r--   1 mickeymo   16 Jul  3 17:39 linuxChap1.doc
-rw-r--r--   1 mickeymo   17 Jul  3 17:39 linuxChap2.doc
-rw-r--r--   1 mickeymo    9 Jul  3 17:39 linuxCont.doc
drwxr-xr-x   2 mickeymo 4096 Jul  3 17:40 linuxInternals
-rw-r--r--   1 mickeymo    7 Jul  3 18:53 listing.lst
```

IBM

Uso de ()

El metacaracter `()` permite agrupar comandos en la línea de comandos

```
[mickeymouse@mycomputer mickeymouse]$ (cat linux.txt; date) > datedfile
[mickeymouse@mycomputer mickeymouse]$
```

`datedfile` contine la salida de **`linux.txt`** y la fecha del sistema cuando se emitió el comando

Uso de && y ||

- `&&` se usa cuando el segundo comando va a ser ejecutado sólo si el primer comando tiene éxito
- `||` se usa cuando el segundo comando va a ser ejecutado sólo si el primer comando falla.
- Cada comando devuelve al shell un estado de salida
- Se devuelve 0 si el comando se ejecuta sin ningún problema (éxito) y distinto de cero si algo va mal
- Este estado de la salida le dice al shell si el comando se ha ejecutado con éxito o no
- En base al estado de la salida, el shell continúa o detiene la ejecución del segundo comando

Sustitución de Parámetros

- El shell proporciona un método para definir y sustituir variables usando metacaracteres de sustitución de parámetros **{}** (corchetes)
- Las siguientes son las cuatro formas posibles de hacerlo:

  ${variable-word} : Muestra el valor de **variable** si esta existe; caso contrario muestra el valor **word**

  ${variable=word} : Muestra el valor de **variable** si esta existe; caso contrario muestra y asigna a **variable** el valor **word**

  ${variable+word} : Si **variable** está asignada, muestra **word**, caso contrario no muestra nada

  ${variable?mesg} : Si **variable** ya está asignada, muestra el valor, caso contrario muestra **mesg** y sale del shell

Uso de Sustitución de Parámetros

```
[mickeymouse@mycomputer mickeymouse]$ echo ${color-green}
green
```

Dado que **color** no está definido se muestra **green**

```
[mickeymouse@mycomputer mickeymouse]$ color=blue
[mickeymouse@mycomputer mickeymouse]$ echo ${color-orange}
blue
```

Muestra **blue** dado que **color** ya está definido. En ambos casos, la variable **color** permanece sin cambios.

```
[mickeymouse@mycomputer mickeymouse]$ echo ${newcolor=green}
green
```

Muestra **green** y le asigna **newcolor** el valor **green**.

Uso de Sustitución de Parámetros… 2

```
[mickeymouse@mycomputer mickeymouse]$ echo ${newcolor+yellow}
yellow
```

Como **newcolor** está asignado, se muestra **yellow. newcolor** no se modificado. Si `newcolor` no está asignado, entonces no se muestra nada.

```
[mickeymouse@mycomputer mickeymouse]$ echo ${newcolor?not defined}
green
```

Como la variable **newcolor** tiene valor asignado, muestra el valor de **newcolor**.

```
[mickeymouse@mycomputer mickeymouse]$ echo ${other?not defined}
bash: other: not defined
```

Escapes y Comillas

- El shell tiene significados especiales para todos los metacaracteres que reconoce.
- Si se les quiere usar con su significado original, ya que el intérprete los entiende de forma diferente se neutralizan usando Escapes y Comillas
- Los tres metacaracteres usados para neutralizar son:
  - \ (Escape)
  - ' ' (Comillas Simples)
  - " " (Comillas Dobles)

IBM

Escape

Sin usar el metacaracter Escape

```
[mickeymouse@mycomputer mickeymouse]$ echo Enter * now
Enter lint.txt  linux.txt  linux1.txt  linux2.doc
linuxChap1.doc  linuxChap2.doc  linuxCont.doc
linuxInternals  listing.lst  mist.doc  nest.lst
unixInternals now
```

Luego de usar el metacaracter Escape

```
[mickeymouse@mycomputer mickeymouse]$ echo Enter \* now
Enter * now
```

Se ha neutralizado el metacaracter ***** ingresando **** inmediatamente antes del *****

Comillas Simples ´´

```
[mickeymouse@mycomputer mickeymouse]$ echo 'Let us get $100
* 20'

Let us get $100 * 20

[mickeymouse@mycomputer mickeymouse]$
```

En esta cadena, dos metacaracteres, **$** y ***** se ven independientemente de su significado especial usando ‘’

```
[mickeymouse@mycomputer mickeymouse]$ echo Let us get \$100
\* 20

Let us get $100 * 20

[mickeymouse@mycomputer mickeymouse]$
```

Aquí se han usado dos metacaracteres **** para anular el significado de los símbolos especiales **$** y *****

Comillas Dobles ""

El uso de las comillas dobles se explica mejor con el comando **date**.

```
[mickeymouse@mycomputer mickeymouse]$ echo date
date
[mickeymouse@mycomputer mickeymouse]$
```

El comando **echo** imprime la cadena date y no la salida del comando **date.**

```
[mickeymouse@mycomputer mickeymouse]$ echo `date`
Sat Mar 2 13:10:28 GMT 2002
[mickeymouse@mycomputer mickeymouse]$
```

La salida del comando **date** se muestra aquí.

```
[mickeymouse@mycomputer mickeymouse]$ echo '*Date = date`'
*Date = `date`
[mickeymouse@mycomputer mickeymouse]$
```

IBM

Comillas Dobles ""

```
[mickeymouse@mycomputer mickeymouse]$ echo "*Date = `date`"
*Date = Sat Mar 2 13:15:30 GMT 2002
```

Se obtiene la cadena *Date, impresa junto a la salida del comando **date**

Las comillas dobles ("") desactivan el significado especial de todos los metacaracteres excepto **$**, **`** y ****

Completar Nombres de Archivos

La capacidad de digitar algunos de los primeros caracteres (uno o más) del nombre del archivo y que siguiendo los caracteres con la tecla tab, el shell complete el resto de los caracteres del nombre del archivo se llama ***Completar Nombres de Archivo***

Esto es útil cuando:

- Cuando los nombres de archivo son muy largos
- Cuando hay un gran número de archivos en un directorio
- Cuando un gran número de archivos tiene nombres similares

Resumen

- Se presentó una visión general del shell de Linux
- Se discutió acerca de las variables shell
- Se describieron los metacaracteres
- Se explicó cómo establecer y referenciar las variables shell

Laboratorio
Fundamentos del Shell

Características del Shell

Objetivos de Aprendizaje

- Explicar las variables predefinidas del shell
- Explicar los parámetros posicionales
- Describir cómo trabaja el comando **`expr`**
- Discutir acerca de los comandos **`read`** y **`test`**
- Describir la familia de filtros **`grep`** y el filtro **`sed`**

Variables Shell Predefinidas

- Para cada shell el sistema define las variables shell predefinidas, a diferencia de las variables definidas por el usuario
- Las variables shell, tanto las predefinidas como las definidas por el usuario, generalmente se pueden fijar de dos formas:
  - ✓ Asignando un valor a la variable: `variable=value`
  - ✓ Activando la variable: `set variable`

IBM

Uso de Algunas Variables Shell

```
[mickeymouse@mycomputer mickeymouse]$ echo $HOME
/home/mickeymouse
[mickeymouse@mycomputer mickeymouse]$
```

En este ejemplo, la salida de echo **$HOME** es la misma que habría sido para **pwd** ya que el directorio actual de trabajo es el directorio **home**.

Nota: **pwd** no es lo mismo que **PWD**.

pwd es el comando y **PWD** es la variable shell

Uso de Algunas Variables Shell... 2

```
[mickeymouse@mycomputer mickeymouse]$ echo $UID
501
[mickeymouse@mycomputer mickeymouse]$
```

501 es el **ID** al que el usuario ha sido asignado cuando se creó el login para el usuario

```
[mickeymouse@mycomputer mickeymouse]$ PS1=`date`$
Sat Jul 7 15:47:45 GMT 2001$
```

Uso de Algunas Variables Shell... 3

¿Qué le ha ocurrido al prompt de comandos

```
[mickeymouse@mycomputer mickeymouse]$?
```

Se ha cambiado la cadena del prompt de

```
[mickeymouse@mycomputer mickeymouse]$
```

a

```
Sat Jul 7 15:47:45 GMT 2001$
```

asignando el resultado del comando **`date`**.

`PS1` es una variable shell que permite a los usuarios cambiar la cadena del prompt a cualquier otra cadena que se desee.

Se ha asignado la salida del comando **`date`** seguido por **`$`** a **`PS1`**.

Nota: Encerrando la fecha dentro de comillas invertidas se consigue la salida del comando **`date`**.

Parámetros Posicionales

- Cada shell proporciona un lenguaje llamado *script shell* o *programa shell*, el cual se puede usar para escribir programas
- El archivo que maneja el script shell se denomina *shell file (archivo shell)*
- Se pueden pasar argumentos de la línea de comandos a un archivo shell.
- Los argumentos están contenidos en variables shell y se les refiere como parámetros posicionales
- Los parámetros posicionales son también conocidos como *variables de argumento*

Características de los Parámetros Posicionales

- Los Parámetros Posicionales están referidos por las variables **`$0,$1,$2,`** etc.
- **`$0`** que contienen el nombre del script, el primer argumento está contenido en **`$1`**, el segundo en **`$2`**, y así sucesivamente
- Los nombres de las variables no son 1, 2, 3 y etc. son sólo notaciones compactas para indicar los parámetros posicionales de un archivo shell
- El dígito siguiente al símbolo **`$`** indica la posición en la línea de comandos
- A diferencia de las variables de un lenguaje de programación regular, los parámetros posicionales no pueden ser alterados

El Comando expr

```
/home/mickeymouse$ expr 2 + 3 + 5
10
/home/mickeymouse$
```

Los operadores que son permitidos en las expresiones proporcionadas para `expr` son **`|`**, **`&`**, **`!=`**, **`=`**, **`<`**, **`<=`**, **`>`**, **`>=`**, **`+`**, **`-`**, **`*`**, **`/`**, y **`%`**.

Como **`*`**, **`>`** y **`<`** son metacaracteres, cuando estos operadores se usan en el comando **`expr`**, deben ser combinados con el carácter \ para retirar el significado especial de estos caracteres.

Ejemplo:

```
/home/mickeymouse$ expr 2 \* 3
6
/home/mickeymouse$
```

Operadores para expr y Descripciones

Expresiones	Descripción
match STRING REGEXP	Compara la cadena STRING con una expresión regular o con otra cadena .
substr STRING POS LENGTH	Proporciona la subcadena de **STRING**. La posición **POS** se cuenta a partir de **1**.
index STRING CHARS	Proporciona el índice en **STRING** donde cualquier **CHARS** es encontrado. Si no, devuelve 0.
length STRING	Proporciona la longitud de **STRING.**
+ TOKEN	Interpreta **TOKEN** como cadena incluso si se usan las palabras como **'match'** o el operador **'/'**.

IBM

El Comando find

- El comando **`find`** se usa para buscar archivos en la jerarquía de directorios
- La sintaxis del comando es:

```
find [ruta] expresión
```

  - **`ruta`** especifica el directorio a partir del cuál debe comenzar la búsqueda
  - Si no se especifica **`ruta`**, se busca en el directorio de trabajo actual

El Comando read

- El shell proporciona un comando **`read`** incorporado
- El comando **`read`** lee una línea de texto desde la entrada estándar y asigna el valor (en otras palabras, el texto) a una variable con nombre
- Sin embargo, el valor del texto asignado no tiene el carácter de nueva línea
- El comando **`read`** se usa básicamente para configurar variables del entorno en **`.bashrc`**

El Comando test

El comando **`test`** en Linux se usa para realizar las siguientes funciones:

- Verificar si el argumento dado es un archivo o un directorio
- Descubrir si se han pasado suficientes argumentos para los scripts shell
- Comparar cadenas o números dados
- El comando **`test`** simplemente devuelve un estado de la salida al shell, reportando éxito o error

Filtros en Linux

- Una expresión regular es la descripción de una plantilla, la cual se usa para hacer coincidir con una cadena
- Las expresiones regulares son fórmulas que hacen coincidir una cadena con un patrón dado
- El patrón usa caracteres que tienen un significado especial
- La expresión regular se especifica como cadena
- Las expresiones regulares se usan para búsquedas de contexto sensitivas y modificación de texto
- Proporcionan un método para seleccionar una cadena específica de un conjunto de cadenas

Algunos Metacaracteres Simples

Carácter Especial	Significado
.	Hace coincidir cualquier carácter único. **`r.n`** coincidirá con **`run`**, **`ran`**, **`rbn`**, **`ron`** etc. **Nota:** El . indica un solo carácter entre **`r`** y **`n`**. La línea puede tener caracteres antes de **`r`** y después de **`n`**. Así, el patrón relaciona **`abrun`**, **`ranbb`** o **`abrankk`**
`[]`	Hace coincidir cualquier carácter especificado en el rango. **`[1234]book`** coincidirá con **`1book`**, **`2book`**, **`3book`** o **`4book`**
`[^]`	Relaciona cualquier carácter que no esté en el rango. [**`^abc`**] coincidirá con todas las líneas que contienen cualquier un caracter que no sea [**`a,b,c`**]. Así, **`adf`** y **`kkkkc`** cumplen, pero no **`aabc`** ó **`aaaa`**

Algunos Metacaracteres Simples... 2

Carácter Especial	Significado
`?`	Hace coincidir cero o una ocurrencia del carácter precedente. **`ab?`** Coincidirá con **`a`** y **`ab`**.
`*`	Hace coincidir cero o más ocurrencias del carácter precedente. **`ab*`** coincidirá con **`a`**, **`ab`**, **`abb`**, **`abbbbb`**, **`aaa`**, **`acc`**, etc. **`aaa`** y **`acc`** también coinciden porque el * relaciona cero o más ocurrencias del carácter precedente, que es **`b`** en este caso
`+`	Hace coincidir una o más ocurrencias del carácter precedente. **`ab+`** coincide con **`ab`**, **`abb`**, **`abbbb`**. etc. No coincide con **`acc`** o **`aaa`** ya que + indica una o más ocurrencias del carácter precedente **`b`**.

Algunos Metacaracteres Simples... 3

Carácter Especial	Significado
`^`	Hace coincidir el inicio de la línea. Por ejemplo, **`^this`** coincidirá con todas las líneas que tienen this como patrón de inicio en las líneas
`$`	Hace coincidir el final de la línea. Por ejemplo, **`this$`** coincidirá con todas las líneas que tienen this como patrón de fin en todas las líneas
`+`	Hace coincidir una o más ocurrencias del carácter precedente. **`ab+`** coincidirá con **`ab`**, **`abb`**, **`abbbb`** etc. No coincidirá con **`acc`** o **`aaa`** ya que **`+`** indica una o más ocurrencias del carácter precedente **`b`**.

grep

- **grep** viene de impresión de expresiones regulares globales (***g****lobal* ***r****egular* ***e****xpression* ***p****rint*)
- El uso más simple de **grep** es:

```
grep pattern filename
```

- El patrón para **grep** pueden ser los metacaracteres **^** y **$**, que se usan para fijar el patrón al inicio o al final de la línea, respectivamente

grep

- Se utilizará el siguiente archivo para los ejemplos de grep :

  filters are programs that read input, process the input and write some output. Some examples of filters

  are sort, tail, head and grep. There are many more

  filters in Linux. grep is a powerful filter that finds the

  pattern in the input and writes to the standard output.

grep – Uso del Comando

```
/home/mickeymouse$ grep filters filters.txt
```

filters are programs that read input, process the input and write some output. Some examples of filters

filters in Linux. grep is a powerful filter that finds the

```
/home/mickeymouse$
```

El uso anterior de **`grep`** es el más simple.

Encuentra el patrón filters en el archivo **`filters.txt`** y muestra las líneas correspondientes

grep – Uso del Comando... 2

```
/home/mickeymouse$ grep ^filters filters.txt
```

filters are programs that read input, process the input and write some output. Some examples of filters

```
/home/mickeymouse$
```

- Muestra sólo las líneas que tienen el patrón filters al principio de la línea.

```
/home/mickeymouse$ grep filters$ filters.txt
```

write some output. Some examples of filters

```
/home/mickeymouse$
```

- Muestra las líneas que tienen el patrón filters al final de la línea

grep – Uso de Comando... 3

```
/home/mickeymouse$ ls -l | grep '^d'
```

- Imprime los nombres de los subdirectorios del directorio de trabajo actual.
- Los nombres de los subdirectorios se obtienen como resultado del comando anterior, como el listado largo muestra los permisos de los archivos y si el archivo es un directorio, el listado contiene el carácter **d**.
- El comando imprimirá las líneas que contienen el carácter '**d**'.
- Las comillas simples se usan para quitar el significado especial de **^**.
- También se puede usar **[list]** y **[^list]** en **grep**.
- Ejemplo:

```
/home/mickeymouse$ ls -l | grep [5-6]
```

  – Hace coincidir todas las líneas que tienen 5 ó 6 en ellas. Las otras líneas son ignoradas.

grep – Uso de Comando... 4

- El punto (.) equivale a un solo carácter.
- Se aplica al carácter previo o metacaracter de la expresión y hace coincidir cualquier número de ocurrencias sucesivas del carácter o metacaracter
- Ejemplo:

```
/home/mickeymouse$ ls -l | grep f
```

  - Hace coincidir todas las líneas que tienen al menos una 'f' en ellas e ignora aquellas que no tienen '**f**'.
  - Se obtiene el resultado indicado, si es existe al menos un archivo en el directorio que tiene el carácter **f** en su nombre. En caso contrario, no se muestra nada.

egrep y fgrep

- **`egrep`** y **`fgrep`** son versiones un poco más especializadas de **`grep`**
- Ambas pueden aceptar un archivo de comando con la opción **`-f`**
- **`egrep`** puede usar los operadores **`|`** y **`( )`** para agrupar expresiones
- La asociación de patrones que usan **`egrep`** y **`fgrep`** son más rápida, ya que la búsqueda se lleva a cabo en paralelo

egrep y fgrep... 2

La asociación de patrones usando **egrep** y **fgrep** se hace de la siguiente forma:

- Usando paréntesis en **egrep**, se puede tener un patrón **(ab)*** que hace coincidir **ab**, **abab**, **ababab** y así sucesivamente
- **egrep** también puede usar **a+** ó **a?**
- **grep** no permite el uso de **+** y **?**
- **a+** hace coincidir una o más ocurrencias de **a**
- **a?** Hace coincidir cero o una ocurrencia de **a**
- **a*** en **grep**, **egrep,** y **fgrep** se refiere a cero o más ocurrencias de **a**

egrep - Uso del Comando

```
/home/mickeymouse$ egrep '(the)+' lines.txt
```

Muestra todas las líneas del archivo **lines.txt** que tienen al menos un patrón que **the**.

+ establece una o más ocurrencias de **the**

```
/home/mickeymouse$ egrep '(t|s)he+' lines.txt
```

Hace corresponder todas las líneas que tienen **t** o **s** antes del patrón **he**.

El operador **|** se usa para denotar **o**

fgrep - Uso de Comando

Asuma que el archivo **idnos** contiene el número de identificación de los estudiantes como se muestra a continuación.

```
/home/mickeymouse$ cat > idnos
101
102
103
^d
/home/mickeymouse$
```

fgrep - Uso de Comando... 2

El archivo **studentdetails** contiene los siguientes detalles.

```
/home/mickeymouse$ cat > studentdetails
101    John
102    Martina
103    Richard
^d
/home/mickeymouse$
```

Usando **fgrep** se pueden encontrar las líneas que coinciden con los patrones dados en el archivo de entrada.

fgrep - Uso de Comando... 3

```
home/mickeymouse$ fgrep -f idnos studentdetails
101     John
102     Martina
103     Richard
home/mickeymouse$
```

- La opción **-f** le indica a **fgrep** de buscar patrones en un archivo.
- **idnos** contiene una lista de números de identificación de estudiantes de una clase.
- **fgrep** busca las cadenas del patrón de búsqueda en el archivo **idnos**.
- Usando los patrones en el archivo **idnos**, **fgrep** busca en el archivo **studentdetails** todos estos patrones

sed

- **sed** es el editor de flujo, derivado del editor **ed**.
- **sed** funciona también como un filtro
- La sintaxis de **sed** es:

```
/home/mickeymouse$ sed 'lista de comandos' nombrearchivos
```

- **sed** lee una línea a la vez de los archivos de entrada, aplica la lista de comandos y muestra las líneas en la salida estándar
- **sed** no altera el archivo original
- Por medio de **sed,** se pueden mostrar, eliminar o sustituir las que se desee con otro texto
- El archivo intermedio procesado puede ser redireccionado a otro archivo

sed... 2

Alguna funciones de **`sed`** son:

- Aceptar como entrada una combinación de números de línea y un comando
- Aceptar como entrada una combinación de patrón y comando
- Sustituir una cadena antigua con una nueva

sed – Uso del Comando

```
/home/mickeymouse$ sed '2q' filters.txt
```

Sale después de mostrar dos líneas, ya que 2 indica el número de líneas y **q** es el comando para salir

```
/home/mickeymouse$ sed '$p' filters.txt
```

Muestra todas las líneas.

```
/home/mickeymouse$ sed '/the/!d'filters.txt
```

Borra todas las líneas que no contienen la cadena **the** en el archivo **filters.txt**.

sed – Uso del Comando... 2

```
/home/mickeymouse$ sed '/^ [ \t]*$/d' filters.txt
```

- Borra todas las líneas en blanco del archivo.
- Este es un ejemplo de un patrón y un comando.
- El patrón es un blanco al inicio (`^`), seguido de un blanco o un tab (`[ \t]`), seguido por cualquier número de ocurrencias del metacaracter previo (`*`) hasta el final de la línea (`$`).
- El patrón `^ [ \t]*$` significa una línea que contiene sólo blancos o tabs.
- **`sed`** borra todas estas líneas y muestra el resto del contenido del archivo

sed – Uso de Comando... 3

La forma general de sustitución es:

`/oldstring/newstring/`

`oldstring` y **`newstring`** pueden ser cadenas simples

```
/home/mickeymouse$ sed 's/filter/Filter/g' filters.txt
```

- ✓ Reemplaza todas las ocurrencias de la cadena anterior **`filter`** con la nueva cadena **`Filter`**.
- ✓ El comando **`g`** representa global.
- ✓ **`g`** reemplaza todas las ocurrencias en la línea. En otro caso, sólo se reemplaza la primera ocurrencia en la línea.
- ✓ El comando **`s`** le indica a **`sed`** de realizar la sustitución.

sed – Uso del Comando... 4

- ✓ **oldstring** y **newstring** también pueden incluir caracteres especiales.
- ✓ El patrón para **oldstring** puede incluir todos los caracteres especiales que incluye **grep**.
- ✓ **newstring** también incluye algunos caracteres especiales. El más útil es **&**, que se activa para todo **oldstring**

```
/home/mickeymouse$ sed 's/Filter/"&"/g' filter.txt
```

Reemplaza todas las ocurrencias de **Filter** por "**filter**" (encerrado dentro de comillas dobles).

sed – Uso del Comando... 5

sed puede tomar las opciones **-n**, **-e**, y **-f**.

- La opción **-n** suprime la impresión automática de las líneas procesadas.
- **sed** automáticamente imprime todas las líneas procesadas.
- **-n** detiene eso y **sed** imprime sólo aquellas líneas afectadas por el comando **p**

```
/home/mickeymouse$ sed -n '$p' filters.txt
```

- Imprime sólo la última línea del archivo.
- La opción **-f** permite a **sed** tomar los comandos de edición de un archivo, al igual que lo hacen **egrep** y **fgrep**.
- La opción **-e** permite mezclar un comando de edición de línea con comandos en un archivo

Resumen

- Se explicaron las variables predefinidas del shell
- Se discutió acerca de los parámetros posicionales
- Se explicó cómo trabaja el comando **`expr`**
- Se discutió acerca de los comandos **`read`** y **`test`**
- Se describió la familia de filtros **`grep`** y el filtro **`sed`**

Laboratorio
Características del Shell

Procesos en Linux

Objetivos de Aprendizaje

- Definir un proceso y sus usos en un sistema operativo
- Discutir el comando **`ps`**
- Discutir los procesos padre e hijo
- Explicar el concepto de procesos de primer plano y segundo plano
- Discutir acerca de cómo administrar el ambiente de procesos
- Discutir el proceso demonio
- Explicar acerca de señales y su importancia en Linux

Concepto de un Proceso

- Un proceso es la unidad básica programada en un procesador por el sistema operativo
- Un proceso es una entidad dinámica
- Cada proceso se ejecuta independientemente de otros procesos en el sistema
- Los procesos interactúan con otros procesos a través de un mecanismo llamado Comunicación Entre Procesos (Inter Process Communication-IPC)
- Cuando los procesos comparten datos, el sistema operativo usa el principio de sincronización para asegurar que se comparten en forma correcta

El Comando ps

- Se puede conocer el estado de un proceso usando el comando **`ps`**
- **`ps`** viene del inglés *process status* (estado de proceso)
- **`PID`** viene de *process identification number* (*número de identifcación del proceso)*.
- Cada proceso obtiene un número de identificación único asignado por el kernel.
- **`TTY`** es el *tipo de terminal* asociado con el proceso. También puede ser otro dispositivo de entrada conectado al sistema
- **`TIME`** es la cantidad de tiempo que el proceso ha estado en ejecución
- **`CMD`** es el nombre del comando que está siendo ejecutado, cuyo reporte de estado se está visualizando, **`bash`** es el proceso shell

Proceso Padre y Proceso Hijo

- El comando **`bash`** es el proceso shell que se está ejecutando
- El proceso shell es llamado el **proceso padre**
- Todos los comandos ejecutados en el shell son procesos hijo del proceso shell
- Los procesos también están organizados en una jerarquía

El Proceso Shell

- Cuando un usuario se conecta, el kernel inicia un proceso shell para el usuario
- Cuando el usuario ingresa un comando, el shell usa el kernel para iniciar un proceso hijo
- El control regresa al shell una vez que el comando se ejecuta
- El comando **`kill`** se usa para terminar el proceso
- Si el comando **`kill`** no puede terminar el proceso, un administrador o el propietario del proceso puede usar la opción -9 con el comando **`kill`**

Procesos en Primer Plano

- Cuando se ingresa cualquier comando en el prompt del shell, se crea un nuevo proceso shell
- En el ejemplo de cal, el comando cal será ejecutado bajo este nuevo shell
- Este proceso de **`cal`** se estará ejecutando en primer plano
- El prompt $ no se mostrará hasta que el programa se complete
- Después de la ejecución, termina el shell recién creado y el shell padre continúa

Procesos en Segundo Plano

- En muchas situaciones es muy útil ejecutar procesos en segundo plano
- El único cambio que se debe hacer es agregar el símbolo **`&`** al final del comando
- Si se toma el ejemplo de **`myprogram`**, al incluir el símbolo &, se mostrará inmediatamente el prompt de comandos después de mostrar el PID del proceso que se ejecuta en segundo plano

Filtros y Procesos en Segundo Plano

```
/home/mickeymouse$ cat myFile | wc &
[1] 1319
/home/mickeymouse$
```

- Se crean dos procesos para cat y wc que se ejecutan en segundo plano
- Sólo se imprime un PID, del último proceso en la secuencia de línea de comandos, el cual es **wc**
- Para terminar un proceso en segundo plano se indica:

```
/home/mickeymouse$ kill -9 1319
/home/mickeymouse$
```

Filtros y Procesos en Segundo Plano... 2

- Para mantener un programa ejecutándose aún luego de desconectarse, se puede usar el comando **`nohup`**
- Llamar a **`nohup`** automáticamente llamará a **`nice`** para hacer su trabajo de que el proceso se ejecute con baja prioridad
- El comando **`at`** permite al usuario iniciar un programa en un momento particular. Este comando se ejecutará como un proceso en segundo plano en el momento acordado

Variables de Entorno

- Al anteponer una variable con un **`$`** se obtiene el valor de las variables
- Las variables que se usarán son **`?`**, **`$`** y **`!`** que son las variables del entorno
- Las variables del entorno son variables que son mantenidas por el shell
- Normalmente se usan para configurar programas utilitarios bajo un sistema como **`lpr`** (se refiere a la impresión fuera de línea) y **`mail`**
- Establecer estas variables como variables de entorno implica que no se tiene que establecer ciertas opciones cada vez que se ingresa al sistema

Administrar el Ambiente de Procesos

- Linux proporciona un alto grado de flexibilidad para ajustar el ambiente de procesos a las necesidades de cada uno
- Algunas de las características son:
  - Cambiar los caracteres Erase y Kill
  - Cambiar el valor de PATH
  - Configurar el tipo de Terminal
  - Crear Accesos Directos (Shortcuts)
  - Exportar la Configuración Personalizada a un Proceso Shell hijo

Cambiar los Caracteres Erase y Kill

- Un carácter erase elimina el carácter de la pantalla
- El carácter por defecto proporcionado por el sistema es **`^X`**
- Se puede cambiar el valor por defecto para usar cualquier otro carácter usando el comando **`stty`** erase **`^E`**, donde **`^E`** es el carácter que se elige
- El carácter kill elimina la línea ingresada
- El carácter kill por defecto es **`^U`**
- Se puede personalizar a un carácter de la preferencia del usuario

Cambiar la Variable PATH

- PATH controla donde busca el shell los comandos a ejecutar
- La secuencia de directorios donde se busca se llama la ruta de búsqueda y se almacena en **`PATH`**
- La secuencia y contenidos de la variable **`PATH`** pueden ser alterados

Configurar el Tipo de Terminal

- Los programas como **`vi`** requieren que el tipo de terminal esté configurado correctamente
- La variable shell **`TERM`** puede usarse para este propósito
- Algunos tipos de terminales son:
  - ansi
  - adm5
  - vt100
- Los tipos de terminales más usados son **`ansi`** y **`vt100`**

Crear Accesos Directos (Shortcuts)

- Es posible tener accesos directos o abreviaturas de comandos, nombres de archivos y nombres de directorios
- Por ejemplo, se puede tener que usar repetidamente el nombre de directorio como

```
/home/mickeymouse/cprograms/projects/share
```

- Se puede establecer

```
/home/mickeymouse$ rep=/home/mickeymouse/cprograms/
                   projects/share
/home/mickeymouse$
```

- Esto permite indicar **`cd rep`** para cambiar al directorio. Así, cuando ingresa

```
/home/mickeymouse$ cd rep
/home/mickeymouse/cprograms/projects/share$
```

El directorio **`share`** se convierte en el directorio de trabajo actual

Exportar la Configuración Personalizada a un Proceso Shell Hijo

- Se usa el comando **`export`** para asegurar que las configuraciones que crea el shell padre están presentes en todos los procesos shell hijo

- Al usar el comando **`export`** sin ningún argumento se lista el conjunto de todos las variables shell que están actualmente exportadas a un proceso shell hijo

Proceso Demonio

- Un programa que corre en segundo plano de comienzo a fin y atiende un requerimiento legal se denomina proceso demonio
- Algunos procesos que corren como procesos demonios son:
  - Demonio de la cola de impresión
  - Demonio del listener de red TCP
  - Demonio del listener de correos
- Los tipos de demonios que se ejecutan en un sistema son específicos al sistema
- Los demonios no se ejecutan debido a que los inicia al usuario, ni tampoco están asociados con un terminal

Señales

- Un proceso en Linux puede recibir diferentes tipos de señales
- Las señales pueden ser capturadas en una aplicación de manera que se puede iniciar una forma limpia para manejarlas
- Linux proporciona un comando llamado **`trap`** que hace esto
- La sintaxis de **`trap`** es:

```
trap command signal-list
```

- Las interrupciones a un proceso manejadas de esta forma asegura una salida limpia del proceso

Resumen

- Se definió qué es un proceso y sus usos en un sistema operativo
- Se explicó el comando **`ps`**
- Se explicaron los procesos padre e hijo
- Se discutió el concepto de procesos de primer plano y segundo plano
- Se explicó cómo administrar el ambiente de procesos
- Se explicó el proceso demonio
- Se discutió acerca de señales y su importancia en Linux

Personalizar el Ambiente de Usuario

Objetivos de Aprendizaje

- Personalizar las variables de entorno en una instalación Linux
- Explicar la funcionalidad de los archivos **`.bash_profile`** y **`.bashrc`**
- Describir un alias y sus usos
- Explicar la historia de comandos

Configuraciones Personalizables

- La personalización del ambiente es un paso más allá de administrar el ambiente de procesos
- Personalizar el ambiente de procesos es una forma simple y potente de lograr un ambiente de trabajo personalizado
- Un ambiente de trabajo personalizado puede lograrse modificando las variables shell

Configuraciones Personalizables... 2

Se aprenderá a personalizar:

- Configuraciones de Ambiente
- Scripts de arranque del Shell
- Alias
- Historia de Comandos

IBM

Configuraciones de Ambiente

- Cada programa se ejecuta en un ambiente
- El shell en el que el programa se está ejecutando define ese ambiente
- El ambiente existe dentro del shell
- El ambiente está definido por los valores asignados a las variables del ambiente
- El comando **`export`** se usa para agregar o modificar variables del entorno

El Comando export

- En Linux, un shell puede ejecutarse dentro de otro shell
- Las variables declaradas en un shell son locales a ese shell y están disponibles sólo dentro de ese shell
- Un shell hijo (otro shell) se ejecuta dentro de un shell padre (el shell que crea otro shell)
- Si el shell hijo necesita usar una de las variables definidas para el shell padre, esa variable debe ser exportada al shell hijo
- Simplemente, invocando **`bash`** en el prompt se puede correr un nuevo shell

IBM

El Comando export... 2

```
[mickeymouse@mycomputer mickeymouse]$ color=blue
[mickeymouse@mycomputer mickeymouse]$ export color
[mickeymouse@mycomputer mickeymouse]$ bash
[mickeymouse@mycomputer mickeymouse]$ echo $color
blue
[mickeymouse@mycomputer mickeymouse]$
```

IBM

El Comando export... 3

- Las variables exportadas se denominan variables globales, ya que son visibles a los otros shells creados dentro de un shell
- El símbolo **$** se usa para evaluar variables de entorno, como con las variables shell,
- El uso del símbolo **$** para evaluar la variable de entorno se hace sólo en el contexto del shell, mientras el shell está interpretando
- El shell realiza la interpretación cuando los comandos se están ingresando en el prompt o cuando bash lee los comandos desde un archivo como **.bashrc** o **.bash_profile**

El Archivo .bash_profile

- Los archivos que empiezan con . (punto) son archivos especiales que usa el sistema para diferentes propósitos
- **`.bash_profile`** es uno de esos archivos
- Cada directorio home tiene una copia del archivo **`.bash_profile`** que se usa para personalizar la configuración del usuario
- **`.bash_profile`** se ejecuta sólo una vez cuando el usuario se conecta al sistema
- En cada inicio de sesión del usuario se tendrá que ejecutar el archivo **`.bash profile`**

El Archivo .bash_profile... 2

Asuma que la entrada en **.bash_profile** es:

```
# .bash_profile
# Obtiene los alias y funciones
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
# Entorno específico del usuario y programas de inicio
PATH=$PATH:$HOME/bin
export PATH
unset USERNAME
```

IBM

El Archivo .bash_profile... 3

Se van a agregar algunas entradas al archivo .bash_profile

```
# .bash_profile
# Obtiene los alias y funciones
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
# Entorno específico del usuario y programas de inicio
PATH=$PATH:$HOME/bin
export PATH
unset USERNAME
date
PS1=$PWD$
```

IBM

El Archivo .bash_profile... 4

Una vez incluidas las 2 líneas anteriores, se guarda el archivo, se sale del sistema y nuevamente se vuelve a conectar.

```
Welcome to Suse Linux 9.1 (i586)
Kernel 2.6.4-52-default
login: mickeymouse
Password:
Last login: Thu May 30 10:58:58 on tty2
Thu May 30 11:20:16 IST 2005
/home/mickeymouse$
```

El Archivo .bashrc

- Los archivos que tienen el sufijo **`rc`** tienen un significado especial
- **`rc`** son las siglas de 'run control' (control de ejecución)
- Estos archivos permiten que los programas sean configurados de acuerdo a lo que el usuario desea
- **`.bashrc`** se refiere al archivo de configuración especial para el programa **`bash`**
- Cada directorio de inicio tiene una copia del archivo **`.bashrc`**

IBM

El Archivo .bashrc... 2

- En el archivo **.bashrc**, se puede colocar los comandos de shell que se van a ejecutar cada vez que se inicia un nuevo programa
- La entrada por defecto en el archivo **.bashrc** file es:

```
# .bashrc
# Alias y funciones específicas del
# usuario
# Definiciones de origen global
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

El Archivo .bashrc... 3

La nueva entrada en **.bashrc** es:

```
# .bashrc
# Alias y funciones específicas del
# usuario
# Definiciones de origen global

if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
whoami
```

El Archivo .bashrc... 4

- Un punto importante es que cada vez que se ejecuta **bash** en el prompt de comandos, se muestra la identificación del usuario (user id)

```
/home/mickeymouse$ bash
mickeymouse
/home/mickeymouse$
/home/mickeymouse$ bash
mickeymouse
/home/mickeymouse$
```

El Archivo .bashrc... 5

- Si se desconecta y se conecta nuevamente, se encontrará la siguiente impresión en la pantalla

```
Welcome to Suse Linux 9.1 (i586)
Kernel 2.6.4-52-default
login: mickeymouse
Password:
Last login: Thu May 30 12:58:58 on tty2
mickeymouse
Thu May 30 11:52:31 IST 2005
/home/mickeymouse$
```

Más sobre El Archivo .bashrc

- **.bash_profile** se ejecuta sólo una vez que el usuario se conecta, mientras **.bashrc** se ejecuta por cada ejecución de **bash**
- **.bashrc** proporciona una forma por la cual se puede personalizar un programa, por ejemplo **bash**
- Hay algunas variables que son configuradas normalmente en el archivo **.bashrc,** ya que son significativas al iniciar una sesión y no un proceso shell, tal como **HOME** y **USER**
- **.bash_logout** se ejecuta cuando el usuario se desconecta del sistema

El Comando env

Nombre de Variable de Entorno	Descripción	Ejemplo
EDITOR	Establece el editor por defecto. Generalmente **`emacs`** o **`vi`**	EDITOR=emacs EDITOR=vi
HOME	Establece el directorio home (de inicio) del usuario	HOME=/home/mickeymouse
SHELL	Establece el programa shell actual en ejecución	SHELL=/bin/bash
TERM	Establece el tipo de terminal que está siendo usado	TERM=ansi
USER	Establece el nombre del usuario actual	USER=mickeymouse

Salida del Comando env

```
/home/mickeymouse$ env
PWD=/home/mickeymouse
REMOTEHOST=192.168.1.201
HOSTNAME=mycomputer
PVM_RSH=/usr/bin/rsh
QTDIR=/usr/lib/qt-2.3.1
LESSOPEN=|/usr/bin/lesspipe.sh %s
XPVM_ROOT=/usr/share/pvm3/xpvm
KDEDIR=/usr
USER=mickeymouse
```

Salida del Comando env... 2

```
LS_COLORS=
MACHTYPE=i686-suse-linux
MAIL=/var/spool/mail/mickeymouse
INPUTRC=/etc/inputrc
LANG=en_US
LOGNAME=mickeymouse
SHLVL=1
SHELL=/bin/bash
HOSTTYPE=i386
OSTYPE=linux
```

Salida del Comando env... 3

```
HISTSIZE=1000
LAMHELPFILE=/etc/lam/lam-helpfile
PVM_ROOT=/usr/share/pvm3
TERM=ansi
HOME=/home/mickeymouse
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-
askpass
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/
usr/bin:/usr/X11R6/bin:/home/mickeymouse/bin
_=/usr/bin/env
/home/mickeymouse$
```

Shell Scripts

- Un **`bash`** se puede ejecutar como:
  - un shell de inicio (login shell),
  - un shell interactivo o
  - un shell no interactivo
- Un shell scripts o simplemente *scripts* es un determinado conjunto de programas que se ejecutan en secuencia cuando arranca un shell
- Algunos de los shell scripts más usados al iniciar o cerrar una sesión son:

  `.bash_profile`

  `.bashrc`

  `.bash_logout`
- Estos shell scripts están disponibles en **`$HOME`**
- **`.bash_profile`** se ejecuta antes de un **`.bashrc`**

Shell Scripts... 2

Otros archivos del directorio de inicio del usuario son :

- **`.emacs`** : Para el editor **`emacs`**
- **`.exrc`**: Para el editor **`vi`**
- **`.newsrc`**: Para lectores de noticias

Esto es relevante cuando un conjunto de nuevos grupos de noticia se ofrece en la máquina

Comando Alias

- El alias es un comando mediante el cual el sistema Linux permite al usuario dar nombres cortos a los comandos

- Ejemplo:

```
/home/mickeymouse$ alias dt=date
/home/mickeymouse$
```

El comando **date** ha sido abreviado como **dt**. Ahora se puede usar **dt** para mostrar la fecha. Si se ingresa **dt** en el prompt de comandos, se obtiene la salida del comando **date**

```
/home/mickeymouse$ dt
Thu May 30 13:14:47 IST 2001
/home/mickeymouse$
```

IBM

Alias... 2

- El alias no puede tomar una opción
- Las opciones para un comando se darán cuando se crea el alias
- Si se quiere abreviar **`ls -al`**, se puede hacer lo siguiente:

```
/home/mickeymouse$ alias la="ls -al"
```

- El shell trata **`ls`** y **`al`** como dos comandos diferentes.
- También se pueden usar tuberías y filtros en un alias.

Historial de Comandos

- Cada comando que se ingresa se almacena en un archivo llamado **`.bash_history`**
- Este archivo está disponible en el directorio de inicio del usuario.
- Al desconectarse del sistema, los comandos usados se agregan al archivo `.bash_history`
- En cualquier momento, se puede determinar los últimos **`n`** comandos utilizados en el sistema.
- El comando **`history`** seguido de un número, muestra los últimos n comandos utilizados

Historial de Comandos - Ejemplo

```
/home/mickeymouse$ history 10
  325  vi .bash_profile
  326  vi .bashrc
  327  vi .bash_profile
  328  exit
  329  cf
  330  man history
  331  clear
  332  vi .bash_history
  333  history 4
  334  history 10
/home/mickeymouse$
```

IBM

Historial de Comandos – Ejemplo... 2

- Se ejecuta el comando usando:

```
/home/mickeymouse$ !329
cf
This is all about aliasing.
It is an interesting feature in Linux.
I am enjoying learning this.
Are there more such features?
/home/mickeymouse$
```

- Se usa el símbolo **!** antes del número para ejecutar el comando.
- Se muestra el comando, seguido por la salida del comando.
- El número **329** fue **cf**, un alias.

Resumen

- Personalizar las variables de entorno de una instalación Linux
- Explicar la funcionalidad de los archivos **`.bash_profile`** y **`.bashrc`**
- Describir un alias y sus usos
- Explicar la historia de comandos