14.Funciones

La sintaxis general de una función es:

```
nombre () { comandos ; }
```

Una función asocia un nombre a una lista de comandos.

```
lsl() { ls -l }

lspath() {
     OLDIFS="$IFS"
     IFS=:
     for DIR in $PATH do
          echo $DIR
     done
     IFS="$OLDIFS"
}
```

Ahora se puede utilizar esta función de la siguiente forma:

```
$ lspath
/bin
/usr/bin
/sbin
/usr/sbin
/usr/X11R6/bin
/opt/bin
/usr/local/bin
/opt/kde/bin
/opt/mozilla/bin
/opt/qt/bin
```

15.Filtros de Texto

Los scripts son muchas veces utilizados para manipular y formatear la salida de los comandos que ejecutan. A veces son tareas simples como mostrar parte de la salida filtrando ciertas lineas. Sin embargo, la mayoría de las veces el procesamiento requerido es más sofisticado.

En este capítulo se mostrarán algunos comandos que se utilizan para filtrar texto

en scripts y como combinarlos de manera útil.

- `head`
- `tail`
- `grep`
- `sort`
- `uniq`
- `tr`

Los Comandos `head` y `tail`

En el capítulo 3 se vió el uso del comando `cat` para visualizar el contenido de un archivo. Este comando permite ver todo el contenido de un archivo, pero hay oportunidades donde necesitamos más control sobre las líneas visualizadas. Para esto se utilizan los comando `head` y `tail`.

El Comando `head`

La sintaxis básica del comando `head` es

```
head [-n líneas] archivos
```

Donde `archivos` es el conjunto de archivos que se quiere procesar con `head` y `líneas` la cantidad de líneas que se quiere mostrar desde el comienzo del archivo. Si se omite la opción `-n`, por defecto el comando muestra 10 líneas.

Este comando es útil por ejemplo si queremos ver los últimos 5 archivos accedidos el directorio /etc:

```
$ ls -1ut /etc | head 5
group
ld.so.cache
nsswitch.conf
passwd
pacman.conf
```

Si quisiésemos ver los últimos 10:

```
$ ls -1ut /etc | head 10
group
ld.so.cache
nsswitch.conf
passwd
```

```
pacman.conf
inputrc
host.conf
hosts
resolv.conf
localtime
```

El Comando `tail`

La sintaxis básica de `tail` es similar a la de `head`:

```
tail [-n líneas] archivos
```

Donde `archivos` es el conjunto de archivos que se quiere procesar con `tail` y `líneas` la cantidad de líneas que se quiere mostrar desde el final del archivo. Si se omite la opción `-n`, por defecto el comando muestra 10 líneas.

En el ejemplo anterior si quisiésemos ver los archivos menos accedidos tendríamos:

```
$ ls -1ut /etc | tail 5
scsi_id.config
shadow-
slp.spi
start_udev
gnome-vfs-mime-magic
```

La Opción `follow`

Una opción extremadamente útil del comando tail es la opción follow `-f`.

```
tail -f archivo
```

Cuando se especifica la opción `-f`, permite inspeccionar un archivo mientras se escribe a este. Es especialmente para seguir logs de programas.

```
$ tail -f /var/log/errors.log
```

El comando `grep`

El comando `grep` (globally regular expression print) permite encontrar líneas en un archivo que contienen una palabra particular.

Búsqueda de Palabras

La sintaxis básica del comando **grep** es:

```
grep palabra archivos
```

Donde **archivo** es el archivo donde se quiere buscar la **palabra**. El comando **grep** mostrará cada línea del archivo donde aparezca la palabra buscada. Cuando se especifica más de un archivo se precede con el nombre del archivo en donde se encontró.

Como ejemplo, se puede buscar las ocurrencias de **Input** en el archivo **/etc/X11/xorg.conf**:

```
$ grep Input /etc/X11/xorg.conf
    #InputDevice    "Touchpad" "AlwaysCore"
    InputDevice    "Keyboard0" "CoreKeyboard"
    InputDevice    "USBMouse" "CorePointer"
Section "InputDevice"
Section "InputDevice"
Section "InputDevice"
```

Si no se encuentra ninguna ocurrencia la salida es vacía.

Ignorando el case

El comando **grep** funciona por defecto diferenciando mayúsculas de minúsculas, por lo cual la salida de

```
$ grep Input /etc/X11/xorg.conf
```

será distinta de

```
$ grep input /etc/X11/xorg.conf
```

Para ignorar el case de la palabra se utiliza la opción **-i**.

Lectura Desde STDIN

Cuando no se especifica ningún archivo, **grep** busca matchear las líneas que son ingresadas desde **STDIN**. Esto lo hace perfecto para utilizar con pipes.

Por ejemplo, el siguiente comando busca al usuario actual en la salida del comando **who**:

```
$ who | grep $USER
```

```
frepond  :0           2007-10-01 21:58
```

La Opción `-v`

La mayoría de las veces se quiere buscar la líneas que contengan una determinada palabra. Sin embargo, existen veces donde se quiere buscar las líneas que no contengan dicha palabra. Para esto se utiliza la opción `-v`.

```
$ grep -v home /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
mail:x:8:12:mail:/var/spool/mail:/bin/false
nobody:x:99:99:nobody:/:/bin/false
dbus:x:81:81:System message bus:/:/bin/false
avahi:x:84:84:Avahi daemon:/:/bin/false
hal:x:82:82:HAL daemon:/:/bin/false
```

Un uso común de la opción `-v` es para buscar en la salida del comando `ps`. Por ejemplo si se estuviese buscando todas las instancias de bash corriendo en el sistema, se podría utilizar:

```
$ ps -ef | grep bash
frepond   6229  6208  0 21:58 pts/0    00:00:00 /bin/bash
root      6608  6607  0 22:09 pts/0    00:00:00 -bash
frepond   6724  6208  0 22:25 pts/2    00:00:00 /bin/bash
frepond   7071  6724  0 23:15 pts/2    00:00:00 grep bash
```

La última salida incluye el comando que se acaba de ejecutar. Dado que no es realmente una instancia de bash se puede filtrar utilizando:

```
$ grep -ef | grep bash | grep -v grep
frepond   6229  6208  0 21:58 pts/0    00:00:00 /bin/bash
root      6608  6607  0 22:09 pts/0    00:00:00 -bash
frepond   6724  6208  0 22:25 pts/2    00:00:00 /bin/bash
```

Números de Línea

Mientras `grep` busca en un archivo determinada palabra, mantiene registro del número de línea examinado. Se puede hacer que el comando imprima el número de línea especificando la opción `-n`.

```
$ grep -n Input /etc/X11/xorg.conf
6:    #InputDevice    "Touchpad" "AlwaysCore"
9:    InputDevice    "Keyboard0" "CoreKeyboard"
```

```
10:    InputDevice    "USBMouse" "CorePointer"
27:Section "InputDevice"
38:Section "InputDevice"
59:Section "InputDevice"
```

Listado Sólo del Nombre de Archivo

Existen veces donde no interesa realmente donde se encontró determinada palabra, sino sólo el archivo que la contiene. Para esto `grep` provee la opción `-l`:

```
$ grep -l error /var/log/*
/var/log/Xorg.0.log
/var/log/Xorg.0.log.old
/var/log/acpid.log
/var/log/acpid.log.1
```

Conteo de Palabras

Contar palabras es una capacidad esencial de un script. Hay muchas maneras de hacerlo, la más fácil es utilizando el comando `wc`. Pero muchas veces este comando se queda corto. Por ejemplo si queremos ver el número de ocurrencias de una palabra en un archivo. Para solucionar este problema se verá el uso de los siguientes comandos:

- `tr`
- `sort`
- `uniq`

El comando `tr` (translate) cambia todos los caracteres de un conjunto por los de otro conjunto o borra caracteres.

El comando `sort` ordena la líneas de un archivo de entrada. Si no se especifica un archivo lee de `STDIN`.

El comando `uniq` (unique) imprime todas la líneas únicas de un archivo. Si una línea aparece varias veces, sólo una copia es impresa. También puede utilizarse para contar la cantidad de veces que una línea está repetida.

El Comando `tr`

Supongamos que queremos ver las palabras más utilizadas en un archivo de texto (generamos uno con `man ls > manls.txt`). Lo primero que se necesita es eliminar los signos de puntuación dado que las palabras `archivo` o `archivo.` son lo mismo. Esto se puede lograr con el comando tr cuya sintaxis básica es:

```
tr 'conjunto1' 'conjunto2'
```

Donde **tr** toma los caracteres del **conjunto1** y los reemplaza por los caracteres del **conjunto2**.

Para realizar esto se utiliza el comando:

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt
```

Aquí se remplazan los signos de puntuación por un espacio, dado que si bien hay que eliminar los signos de puntuación, las palabras deben estar separadas.

A esta altura, la mayoría de las palabras están separadas espacios, para obtener una cuenta más precisa debemos reemplazar las palabras capitalizadas:

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt | tr 'A-Z' 'a-z'
```

Compresión de Caracteres

A este punto uno o más espacio separan las palabras del. Se necesita comprimir (squeeze) los espacios a uno para evitar problemas al contar. Para esto se utiliza la opción **-s**. La sintaxis básica es:

```
tr -s 'set1'
```

Entonces el comando queda:

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt | tr 'A-Z' 'a-z' | tr -s '
'
```

El Comando sort

Para obtener la cuenta de cuantas veces es utilizada una palabra necesitamos utilizar el comando **sort**. En la forma básica, sort ordena cada línea de la entrada. Para es esto aún necesitamos formatear la entrada de **sort** para que cambie los **espacios** por **\n**.

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt | tr 'A-Z' 'a-z' | tr -s '
' | tr ' ' '\n'
```

Ahora tenemos una lista de todas la palabras en líneas ordenadas.

El comando uniq

Ahora se necesita contar la cantidad de ocurrencias de cada palabra. Para esto se utiliza el comando **uniq**. Dado que el comando **uniq** filtra las líneas iguales consecutivas, fue necesario el **sort** anterior. Por ejemplo si utilizamos el comando con la siguiente entrada:

```
$ uniq << END
> ferrari
> ferrari
> audi
> bmw
> bmw
> ferrari
> END
```

La salida obtenida es:

```
ferrari
audi
bmw
ferrari
```

Donde **ferrari** aparece duplicado.

Entonces el comando que cuenta la cantidad de ocurrencias de cada plalabra queda:

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt | tr 'A-Z' 'a-z' | tr -s '
' | tr ' ' '\n' | sort | uniq -c
```

Ordenamiento de Números

A esta altura se tiene la salida ordenada alfabéticamente con la cantidad de ocurrencias de cada palabra. Sería mucho más fácil determinar cuales son la de mayor ocurrencia ordenando la salida por el número de ocurrencias y por ejemplo obteniendo el top 5. Para esto se hace:

```
$ tr '!?":;\[\]{}(),.\t\n' ' ' < manls.txt | tr 'A-Z' 'a-z' | tr -s '
' | tr ' ' '\n' | sort | uniq -c | sort -rn | head -n 5
```

Ordenamiento de Números en una Columna Diferente

Utilizando el comando **sort -rn** se pudo ordenar la salida porque los números aparecían en la primer columna. El comando **sort** permite ordenar aún cuando los valores numéricos no aparecen el la primer columna con la opción **-k**.

La sintaxis general es

```
sort -k comienzo,fin archivos
```

Donde `comienzo` es la columna de comienzo para la clave, y `fin` es la columna de finalización. La primer columna es 1, la segunda 2 y así sucesivamente.

```
$ ls -l | sort -nr -k 5
-rw-r--r-- 1 frepond users 7626331 2007-09-28 21:31 spring-
javaconfig-1.0-m2-with-dependencies.zip
-rw-r--r-- 1 frepond users 5201442 2007-09-28 20:05 sannotations-
base.tar.gz
-rw-r--r-- 1 frepond users 1294336 2007-10-01 18:53 dus.eap
-rw-r--r-- 1 frepond users  561647 2007-09-28 21:30 spring-
javaconfig-1.0-m2.zip
-rw-rw-rw- 1 frepond users   98728 2007-06-10 15:58 spring-
annotation-base-1.1.1.GA.jar
```

Clases de Caracteres con `tr`

A veces especificar los caracteres en forma individual como los caracteres de puntuación del ejemplo es tedioso y propenso a errores. El comando `tr` permite especificar clases de caracteres. La clases de caracteres son un conjunto predefinido de caracteres. Las clases reconocidas por `tr` se muestran en la siguiente tabla.

Clase	Descripción
`alnum`	Letras y dígitos
`alpha`	Letras
`blank`	Espacios en blanco horizontales
`cntrl`	Caracteres de control
`digit`	Dígitos
`graph`	Caracteres imprimibles sin incluir el blanco
`lower`	Caracteres en minúscula
`print`	Caracteres imprimibles incluidos espacios
`punct`	Signos de puntuación
`space`	Caracteres en blanco, horizontales y verticales
`upper`	Letras en mayúsculas
`xdigit`	Dígitos hexadecimales

La forma de invocar tr con clases de caracteres es

```
tr '[:clase:]' 'set2'
```

En el ejemplo anterior quedaría:

```
$ tr '[:upper:]' '[:lower:]' < manls.txt | tr -cd '[:alnum:]_ \n' |
tr -s ' ' '\n' | sort | uniq -c | sort -n -r | head -n 5
```