MANUAL 4

SHELL SCRIPTS

FILTROS

Contenido

1. Expresiones regulares

Una expresión regular es un patrón que define a un conjunto de cadenas de caracteres. Muchos comandos de procesamiento y búsqueda de texto como `grep`, `egrep`, `sed` o `awk` usan expresiones regulares.

Las expresiones regulares suelen construirse a partir de caracteres normales y/o de caracteres especiales, algunas veces llamados metacaracteres.

Hay dos tipos de expresiones regulares:

- Expresiones regulares básicas (ER)
- Expresiones regulares extendidas (ERE)

Las expresiones regulares básicas (ER) y las extendidas (ERE) difieren solo en los metacaracteres que contienen.

IMPORTANTE: Al utilizar expresiones regulares se recomienda siempre usar comillas.

1.1. Expresiones regulares básicas

Comandos como grep o sed usan las expresiones regulares básicas por defecto. Para poder utilizar las expresiones regulares extendidas hay que hacer uso de `egrep` o `grep -E` (en lugar de grep) o de `sed -r` (en lugar de sed)

ER de un sólo carácter

ER ERE	**concuerda con**
`.`	cualquier carácter
`[ ]`	cualquiera de los caracteres entre corchetes, P.e. `[abc]` concuerda con `a`, `b` o `c`; `[a-z]` concuerda con cualquier letra minúscula
`[^ ]`	cualquier carácter que NO esté entre corchetes
`^`	inicio de línea
`$`	final de línea
`*`	0 o más ocurrencias de la expresión regular anterior
`\( \)`	permite agrupar ER
`\`	*escapa* un metacarácter

- Dentro de [ ] los metacaracteres pierden su significado especial: p.e. [a.]c concuerda con ac y .c

Ejemplos:

ER	**concuerda con**
`a..c`	cadena que empiece por `a`, seguida por dos caracteres y `c`: `a00c`, `xaxxcxx`, `aacc`,...
`0[abc]0`	cadenas que tengan un `0` seguido de un carácter `a`, `b`, o `c` y seguido de otro `0`: `0a0`, `00ab0b0`, `bc0c0`,...
`0[^abc]0`	cadenas que tengan un `0` seguido de un carácter distinto a `a`, `b`, o `c` y seguido de otro `0`
`0[a-z]0`	cadenas que tengan un `0` seguido de una letra minúscula, y `0`

`^abc`	líneas que empiecen por `abc`
`abc$`	líneas que terminen por `abc`
`ab*c`	cadenas que empiecen por `a`, que continúen con 0 o más `b`, y una `c`: `abc`, `ac`, `abbc`, `aaccab`,...pero no `cba` o `aaab`
`b[cq]*e`	cadenas que empiecen por `b`, que continúen con 0 o más `c` o `q`, y una `e`: `be`, `bcce`, `bccqqee` o `bqqqce`
`.*`	cualquier cadena
`0\(abc\)*0`	cadenas que tengan un `0` seguido de 0 o más ocurrencias de `abc`, y seguido de otro `0`: `0abc0`, `00`, `0abcabc0`,..., pero no `0ac0` o `0cba0`
`^#.*\.$`	línea que empiece por `#` y termine por `.` (notar que el segundo `.` está *escapado* por la `\`; la ER `.*` implica 0 o más caracteres cualquiera)

Repetición

Podemos repetir una ER usando \{ \}

Constructor	Propósito
`\{n\}`	concuerda con exactamente *n* ocurrencias de la RE previa
`\{n,\}`	concuerda con al menos *n* ocurrencias de la RE previa
`\{n, m\}`	concuerda con entre *n* y *m* ocurrencias de la RE previa

Ejemplos:

- `a\{5\}`: 5 ocurrencias del carácter `a`
- `0\{2,\}`: al menos 2 ocurrencias del carácter `0`

1.2. Expresiones regulares extendidas

Algunos comandos (`egrep`, `grep -E`, `sed -r`) admiten extensiones a las expresiones regulares básicas:

ERE	concuerda con
+	una o más ocurrencias de la RE anterior
?	cero o una ocurrencia de la RE anterior

Además, `\( \)` y `\{ \}` se reemplazan por `( )` y `{ }`

Ejemplos:

- `ab+c` concuerda con `abc`, `abbc`, pero no con `ac`
- `ab?c` concuerda con `ac`, `abc`, pero no con `abbc`

Para usar los caracteres `(`, `)`, `{` o `}` literalmente hay escaparlos con `\`

Alternancia

El carácter | permite alternar entre 2 o más RE

- `(a|b)c` concuerda con `ac` o `bc`

Otros caracteres

Además de los ya vistos, pueden usarse otros metacaracteres:

ER	concuerda con
`[:space:]`	caracteres en blanco ([\t\n\r\f\v])
`[:blank:]`	espacio y tabulado
`[:alnum:]`	caracteres alfánuméricos (letras y números)
`[:digit:]`	dígitos
`[:alpha:]`	alfabéticos
`[:upper:]`	mayúsculas
`[:lower:]`	minúsculas
`[:xdigit:]`	dígitos hexadecimales
`[:punct:]`	signos de puntuación
`[:cntrl:]`	caracteres de control
`[:graph:]`	caracteres imprimibles (sin espacio)
`[:print:]`	caracteres imprimibles (con espacio)
`\<, \>`	inicio/fin de palabra

2. Filtros

Los filtros son comandos que reciben una entrada, la procesan y escriben una salida. Son muy utilizados en la programación Shell scripts.

Ejemplos de acciones que llevan a cabo los filtros son:

- Seleccionar un parte de la entrada
- Ordenar la entrada
- Modificar la entrada
- Etc.

2.1. head (ver las primeras líneas de un archivo de texto)

El comando `head` muestra las primeras líneas de un archivo. Si se ejecuta sin opciones muestra las 10 primeras líneas.

Sintaxis:

```
head [opción] [archivo]
```

Opciones más importantes:

`-N`	Imprime las primeras N líneas en vez de 10

`-n N`	Imprime las primeras N líneas en vez de 10
`-c N`	Imprime los primeros N bytes del archivo

Ejemplo: Muestra las primeras 4 líneas del archivo `documento.txt`

```
$ head -4 documento.txt
```

2.2. tail (ver las últimas líneas de un archivo de texto)

El comando `tail` muestra las últimas líneas de un archivo. Si se ejecuta sin opciones muestra las 10 últimas líneas.

Sintaxis:

```
tail [opción] [archivo]
```

Opciones más importantes:

`-N`	Imprime las últimas N líneas del archivo en vez de 10
`-n N`	Imprime las últimas N líneas del archivo en vez de 10
`-n +N`	Imprime todas las líneas menos las N - 1 primeras
`-c N`	Imprime los últimos N bytes del archivo

Ejemplo: Muestra las últimas 4 líneas del archvo `documento.txt`

```
$ tail -4 documento.txt
```

2.3. wc (contar bytes, palabras o líneas de un archivo)

El comando wc imprime el número de bytes, palabras y líneas de un archivo de texto.

Sintaxis:

```
wc [opciones] [archivo]
```

Opciones más importantes:

`-l`	Imprime el número de líneas
`-w`	Imprime el número de palabras
`-c`	Imprime el número de bytes

`-m`	Imprime el número de caracteres
`-L`	Imprime la longitud de la línea más larga

Ejemplo: Muestra el número de líneas que tiene el archivo `script.sh`

```
$ wc -l script.sh
12 script.sh
```

2.4. cut (extraer columnas de un archivo)

El comando `cut` extrae columnas de texto a partir de archivos. Es muy útil, ya que nos permite trabajar con determinadas columnas o campos de un fichero.

Sintaxis:

```
cut -(b|c|f)intervalo [opciones] [archivo]
```

Opciones más importantes:

`-b intervalo`	Selecciona solo esos bytes por línea
`-c intervalo`	Selecciona solo esos caracteres por línea
`-d DELIM`	Usa el carácter DELIM en vez del tabulador para delimitar un campo.
`-f intervalo`	Selecciona solo esos campos/columnas.

La sintaxis de `intervalo` es:

N	Muestra solo ese byte, carácter o columna.
`N-`	Desde N hasta el final de línea
`N-M`	Desde N hasta M (incluido)
`-M`	Desde el primero hasta M (incluido)

Ejemplo: Muestra los cuatro primeros caracteres de cada línea del archivo `/etc/paswd`

```
$ cut -c-4 /etc/passwd
```

Ejemplo: Muestra el primer y cuarto campo del archivo `/etc/paswd`

```
$ cut -d: -f1,4 /etc/passwd
```

2.5. sort (ordenar líneas de texto de acuerdo a diversos criterios)

Se utiliza para ordenar líneas de texto a partir de varios criterios.

Sintaxis:

```
sort [opción] [archivo]
```

El criterio de orden que utiliza sort por defecto es alfabético, esto se debe tener en cuenta siempre que se necesite ordenar listas de números, si no se le específica a `sort` que debe ordenar numéricamente, tomará a los números como una lista de palabras y el resultado no será el deseado. Por ejemplo, alfabéticamente el número 10 está antes que el número 2.

Opciones más importantes:

`-t SEP`	Utiliza SEP como separador. Si no se especifica está opción se utiliza el espacio en blanco o tabulador como separador por defecto.
`-n`	Ordena numéricamente (el 9 antes del 10) en lugar de alfabéticamente (el 10 antes del 9 ya que empieza por 1).
`-r`	Invierte la salida: ordena de mayor a menor.
`-k número`	Especifica la columna o campo sobre la que vamos a realizar la ordenación (combinar con –t para elegir un carácter de separación entre campos)
`-u`	Permite eliminar todas las líneas repetidas después de realizar la ordenación
`-f`	Ordena sin distinción entre mayúsculas y minúsculas.

Ejemplo: Ordenar el archivo /`etc/passwd` por orden alfabético

```
$ sort /etc/passwd
```

Ejemplo: Ordena por campos separados por ":", tomando en cuenta para la comparación los caracteres desde el primero del campo 1 hasta el último del campo 3

```
$ sort -t: -k1,3 /etc/passwd
```

Supongamos el archivo alumnos.txt con las siguientes líneas:

```
Fernando, Santos
Alberto, Vicente
Juan, Domínguez
```

Ejemplo: Ordena el archivo alumnos.txt por el segundo campo (campos separados por ",")

```
$ sort -k2 -t, alumnos.txt
Juan, Domínguez
Fernando, Santos
Alberto, Vicente
```

Ejemplo: Obtener un listado de los ficheros del directorio actual, ordenado de mayor a menor por tamaño de archivo

```
$ ls -l | sort -nr -k5
```

Ejemplo: Ordenar numéricamente el fichero /etc/passwd por el campo UID (campo 3)

```
$ sort -n -t: -k3 /etc/passwd
```

2.6 uniq (localizar líneas idénticas en un archivo)

El comando `uniq` se ejecuta en líneas de texto duplicadas consecutivas.

Sintaxis:

```
uniq [opción] [archivo]
```

Por ejemplo, si tiene el archivo `miDoc.txt`:

```
$ cat miDoct.txt
a
b
b
c
b
```

El comando `uniq` detecta y procesa las dos `b` consecutivas, pero no la tercera.

```
$ uniq miDoct.txt
a
b
c
b
```

`uniq` suele usarse tras ordenar un archivo:

```
$ sort miDoct.txt | uniq
a
b
c
```

En este caso, solo se conserva una b ya que las tres se colocaron juntas mediante la ordenación con sort y después uniq no muestra las líneas repetidas.

También puede contar líneas duplicadas en lugar de eliminarlas.

Ejemplo: Cuenta las ocurrencias de cada línea

```
$ sort miDoct.txt | uniq -c

1 a
3 b
1 c
```

Opciones más importantes:

`-c`	Contar líneas duplicadas adyacentes
`-d`	Imprimir solo líneas duplicadas

2.7. tr (cambiar unos caracteres por otros)

El comando tr permite realizar cambios de unos caracteres por otros.

Sintaxis:

```
tr [opciones] conjunto_caracteres_1 [conjunto_caracteres_2]
```

Opciones más importantes:

`-d`	Elimina los caracteres indicados en `conjunto_caracteres_1`
`-s`	Elimina los caracteres duplicados adyacentes de `conjunto_caracteres_1`
`-c`	Convierte todos los caracteres que no estén indicados en `conjunto_caracteres_1` por `conjunto_caracteres_2`
`-t`	Si `conjunto_caracteres_1` es más extenso que `conjunto_caracteres_2`, recorta conjunto_caracteres_1 para que tengan la misma longitud. Si no se usa –t, el último carácter de `conjunto_caracteres_2` se repite hasta que `conjunto_caracteres_2` tenga la misma longitud que `conjunto_caracteres_1`

Ejemplo: (OPCIÓN 1) Convierte todos los caracteres en mayúsculas del archivo /etc/passwd

```
$ cat /etc/passwd | tr 'a-z' 'A-Z'
```

Ejemplo: (OPCIÓN 2) Convierte todos los caracteres en mayúsculas del archivo /etc/passwd

```
$ tr '[:lower:]' '[:upper:]' < /etc/passwd
```

Ejemplo: Convierte todas las vocales por un asterisco

```
$ cat /etc/passwd | tr aeiouAEIOU '*'
```

Ejemplo: Convierte la a y e por A y E, el resto de vocales también las sustituye por E (ya que los conjuntos tienen diferente longitud)

```
$ tr aeiou AE < /etc/passwd
prEpArAdEr:x:1000:1000:prEpArAdEr,,,:/hEmE/prEpArAdEr:/bEn/bAsh
```

Ejemplo: Convierte solo las a y e en A y E por haber utilizado la opción -t

```
$ tr –t aeiou AE < /etc/passwd
prEpArAdor:x:1000:1000:prEpArAdor,,,:/homE/prEpArAdor:/bin/bAsh
```

2.8 find (buscar archivos mediante varios criterios)

El comando `find` explora una rama de directorios buscando archivos que cumplan determinados criterios.

Permite criterios de búsqueda tales como:

- el nombre contiene cierta cadena de caracteres o aparea con algún patrón.
- son enlaces a ciertos archivos.
- fueron usados por última vez en un cierto período de tiempo.
- tienen un tamaño comprendido dentro de cierto intervalo.
- son de cierto tipo (regular, directorio, enlace simbólico, etc.).
- pertenecen a cierto usuario o grupo.
- tienen ciertos permisos de acceso.
- contienen texto que aparea con cierto patrón.

Una vez ubicados los archivos, find puede realizar diversas acciones sobre ellos:

- ver o editar.
- guardar sus nombres en otro archivo.
- eliminar o cambiar de nombre los archivos.
- cambiar sus permisos de acceso.
- ejecutar otras acciones sobre ellos.

Sintaxis:

```
find [ruta..] [expresión]
```

- `ruta`: indica el punto de partida desde donde se deberá iniciar la búsqueda.
- `expresión`: indica un conjunto de opciones como la especificación del archivo a buscar, comparaciones, operaciones y acciones sobre el resultado.

Argumentos numéricos:

`+N`	Valores mayor que N
`-N`	Valores menor que N
`N`	Exactamente N

Criterios de búsqueda más importantes:

`-name PATRON`	Nombre que coincida con PATRON
`-iname PATRON`	Igual que el anterior pero ignorando mayúsculas
`-path PATRON`	Nombre con ruta completa
`-type [dflbc]`	tipo de fichero a buscar d: directorio f: fichero regular l: enlace simbólico b: fichero especial de bloques c: fichero especial de caracteres
`-atime N`	Último acceso al archivo N*24 horas atrás
`-ctime N`	Último cambio de estado al archivo N*24 horas atrás
`-mtime N`	Última modificación al archivo N*24 horas atrás
`-amin N`	Último acceso al archivo N minutos atrás
`-cmin N`	Último cambio de estado al archivo N minutos atrás
`-mmin N`	Última modificación al archivo N minutos atrás
`-size N[bckwMG]`	Tamaño de N unidades de espacio. Las unidades de espacio pueden ser: b (bloques de 512 bytes) c (bytes) w (palabras (2 bytes)) k (KB) M (MB) G (GB)
`-empty`	Archivo vacío (regular o directorio)
`-user UNAME`	Archivo del usuario UNAME

`-group GNAME`	Archivo del grupo GNAME
`-uid N`	Archivo del usuario con UID N
`-gid N`	Archivo del grupo con GID N
`-nouser`	Archivo sin dueño asignado
`-maxdepth N`	Busca de forma recursiva hasta un máximo de N niveles de subdirectorios por debajo del especificado. Por ejemplo, con `-maxdepth 1` se busca en el directorio actual de forma no recursiva (no busca en los subdirectorios). Por defecto si no se indica la opción `-maxdepth` busca en todos los niveles de subdirectorios por debajo del especificado.

Acciones:

`-exec COMANDO {} \;`	Ejecuta COMANDO sobre los archivos encontrados

Ejemplo: Buscar los archivos con extensión .txt en el directorio /home/preparador

```
$ find /home/preparador -name "*.txt"
```

Ejemplo: Buscar en el directorio /home/preparador los archivos con extensión .c con tamaño mayor de 100 K

```
$ find /home/preparador -name "*.c" -size +100K
```

Ejemplo: Buscar en el directorio /home/preparador los archivos que fueron leídos entre 2 y 6 minutos atrás

```
$ find /home/preparador -amin +2 -amin -6
```

Ejemplo: Borrar todos los subdirectorios de /home/preparador que tengan una antigüedad mayor de 10 días;

```
$ find /home/preparador -mtime +10 -type d -exec rm -r {} \;
```

2.9. grep (Buscar en un archivo las líneas que coincidan con una expr. regular)

El comando `grep` recorre uno o varios archivos y muestra todas las líneas de dichos archivos que coincidan con un determinado patrón de expresión regular.

Si no se especifica ningún nombre de archivo, tomará la entrada estándar, por lo que podemos unirlo con tuberías con otros filtros.

Sintaxis:

```
grep [opciones] patrón [archivos]
```

Ejemplo: Imagina un archivo llamado `parque.txt` con las siguientes líneas

En el parque hay mucha gente.

Los niños juegan.

Los adultos vigilan a los niños.

Para buscar todas las líneas del archivo `parque.txt` que contengan la palabra *niños*, se utiliza:

```
$ grep niños parque.txt
Los niños juegan.
Los adultos vigilan a los niños.
```

Opciones más importantes:

`-c`	Imprime la cantidad de líneas que coinciden
`-i`	Ignora la distinción entre mayúsculas y minúsculas
`-l`	Imprime solo los nombres de los ficheros que contengan líneas que coincidan, pero no muestra las líneas
`-n`	Por delante de cada línea que coincida, imprime su número de línea original
`-v`	Imprime solo las líneas que NO coinciden con la expresión regular
`-r`	Busca recursivamente todos los archivos de un directorio y sus subdirectorio
`-E`	Usa expresiones regulares extendidas. Ver `egrep`
`-F`	Usa listas de cadenas fijas en lugar de expresiones regulares. Ver `fgrep`

Ejemplo: Mostrar las líneas que empiecen por la letra `c` en el archivo `misDatos.txt`

```
$ grep '^c' misDatos.txt
```

Ejemplo: Mostrar las líneas que NO comiencen por la letra `c` en el archivo `misDatos.txt`

```
$ grep '^[^c]' misDatos.txt
```

Ejemplo: Mostrar las líneas del archivo /etc/passwd cuyo tercer campo tiene una sola cifra

```
$ grep ^[^:]*:[^:]*:[0-9]: /etc/passwd
```

Ejemplo: Mostrar las cuentas de usuario que NO utilicen bash

```
$ grep –v bash$ /etc/passwd
```

2.9.1. fgrep

El comando `fgrep` es similar a `grep`, pero en lugar de aceptar una expresión regular, acepta una lista de cadenas fijas, separadas por nuevas líneas. Es igual que `grep –F`.

Ejemplo: Buscar las cadenas abanico y alfombra en un archivo llamado archivo.txt

```
$ fgrep 'abanico
alfombra' archivo.txt
```

Cuando a `fgrep` se le proporcionan varios objetivos de búsqueda (dos o más), cada uno debe estar en una línea separada como en el ejemplo anterior.

Para que `fgrep` tome los objetivos de búsqueda desde un archivo en lugar de tenerlos que teclear directamente se utiliza la opción `–f`. Mucho más cómodo cuando hay que buscar varios objetivos.

Ejemplo: Si se tiene un archivo de diccionario con palabras (objetivos de búsqueda), una por línea:

```
$ cat diccionario.txt
abanico
alfombra
barco
…
```

Se pueden buscar dichas cadenas en un conjunto de archivos de entrada:

```
$ fgrep –f diccionario.txt archivo_entrada1 archivo_entrada2
```

2.9.2. egrep

El comando `egrep` es similar a `grep`, pero usa un conjunto más completo y potente de expresiones regulares que `grep`. Es similar a `grep -E`

Ejemplo: Buscar las líneas donde aparezcan alguno de los siguientes nombres: juan, alfonso y Fernando.

```
$ egrep "juan|alfonso|Fernando" alumnos.txt
```

2.10. sed (editor de textos en línea de comandos)

El comando `sed` (Stream EDitor) es un editor de flujos y ficheros orientado a línea de comandos. Este comando recibe por stdin (o vía fichero) una serie de líneas para manipular, y procesa según las opciones especificadas a un rango de las mismas o a las que cumplan alguna condición.

Su funcionamiento se puede resumir de la siguiente forma:

- Paso 1: Lectura de una línea desde el flujo de entrada (las líneas están delimitadas por un carácter de salto de línea (\n))
- Paso 2: La línea se procesa en base a los parámetros especificados
- Paso 3: Muestra el resultado en la salida estándar (pantalla)
- Paso 4: Continúa con la siguiente línea (volver al paso 1)

Sintaxis:

```
sed [opciones] [ámbito]instrucción [fichero]
```

De no indicarse el fichero se aplicarla edición sobre la entrada estándar. Para evitar problemas de interpretación por parte del shell la parte de las sentencias se suele escribir entre comillas dobles o simples. El comando `sed` nunca modifica el flujo de entrada o el texto a editar (la salida modificada se dirige a la pantalla o a un fichero si usamos el operador de redirección).

Opción más importante:

`-n`	No muestra por pantalla las líneas que han sido procesadas
`-r`	Permite usar expresiones regulares extendidas

Ámbito: Muchos de las instrucciones de sed se aplican a un ámbito o contexto que se puede indicar de algunas de las siguientes formas:

- ***num*** = Indica un número de línea
- ***num1, num2*** = Representa un rango de líneas
- ***$*** = Representa la última línea
- ***/regexpr/*** = Todas las líneas que concuerden con la expresión regular regexpr

Instrucciones más importantes para aplicar a `sed`:

`i`	Insertar línea antes de la línea actual
`a`	Insertar línea después de la línea actual.
`c`	Cambiar línea actual
`d`	Borrar línea actual
`p`	Imprimir línea actual en stdout.
`s`	Sustituir cadena en línea actual
`!`	Aplicar instrucción a las líneas no seleccionadas por la condición

Debido a la versatilidad del comando sed los ejemplos se han clasificado en función del tipo de procesamiento que lleva a cabo:

EJEMPLOS DE SUSTITUCIÓN:

Ejemplo: Sustituir `cadena1` por `cadena2` (sustituye solo la primera aparición de la cadena buscada)

```
$ sed 's/cadena1/cadena2' fichero
```

Ejemplo: Sustituir `cadena1` por `cadena2` (sustituye todas las apariciones de la cadena buscada)

```
$ sed 's/cadena1/cadena2/g' fichero
```

Ejemplo: Sustituir la cadena `coche` por `vehículo` sólo en las líneas 5 y 6:

```
$ sed '5,6 s/coche/vehiculo/g' fichero > fichero2
```

EJEMPLOS DE ELIMINACIÓN:

Ejemplo: Eliminar las líneas de la 2 a la 4 de `fichero`:

```
$ sed '2,4d' fichero
```

Ejemplo: Eliminar todas las líneas de `fichero` excepto el rango de la 2 a la 4:

```
$ sed '2,4!d' fichero
```

Ejemplo: Eliminar la última línea de `fichero`

```
$ sed '$d' fichero
```

Ejemplo: Eliminar desde la línea 2 hasta el final de `fichero`:

```
$ sed '2,$d' fichero > fichero2.txt
```

Ejemplo: Eliminar las líneas que contentan la cadena `casa`:

```
$ sed '/casa/d' fichero
```

Ejemplo: Eliminar las líneas que contentan la cadena `casa` al comienzo de la línea:

```
$ sed '/^casa/d' fichero
```

Ejemplo: Eliminar las líneas que contentan únicamente la cadena `casa` en una línea:

```
$ sed '/^casa$/d' fichero
```

Ejemplo: Eliminar líneas en blanco:

```
$ sed '/^$/d'
```

Ejemplo: Eliminar tabuladores o múltiples espacios en blanco dejando un solo espacio (OJO)

```
# sed 's/  */ /g' fichero
```

3. Sustitución de comandos

La sustitución de comandos permite que la salida de un comando reemplace al comando en sí. La sustitución de comandos se produce cuando un comando se utiliza de la siguiente manera:

```
$(comando)
```

o

```
`comando`
```

Ejemplo: Utilizando sustitución de comandos con la sintaxis $()

```
if [ $(id –u alfonso) –ge 1000 ]
then
   comandos
fi
```

Ejemplo: Utilizando sustitución de comandos con la sintaxis de comillas invertidas (` `)

```
if [ `id –u Alfonso` –ge 1000 ]
then
   comandos
fi
```