Línea de comandos: Manipulación de flujos de texto

cat y echo

Estos comandos ya fueron abordados en el primer apunte. Los repasamos brevemente.

El comando cat sirve para mostrar el contenido de un archivo por la salida estándar (o adonde sea redirigida). Su uso es

```
cat nombreDeArchivo
```

El comando echo sirve para mostrar un mensaje por pantalla. Por ejemplo:

```
echo "Buenas noches"
```

mostrará por pantalla el mensaje "Buenas noches".

Puede utilizarse redirigiendo la salida hacia otros archivos. Ejemplo:

```
echo "Buenas noches" > saludo.txt
```

hará que el contenido del archivo "saludo.txt" sea "Buenas noches". Si el archivo existía, su contenido anterior se pierde. Si no existía, será creado. El comando

```
echo "Buenas noches" >> saludo.txt
```

(nótese el doble signo: **>>**) agregará al final del archivo "saludo.txt" la línea "Buenas noches". Si el archivo existía, su contenido anterior se conserva, agregándose la nueva línea. Si no existía, será creado.

cut

En un archivo de texto separado en columnas, este comando muestra las columnas o campos seleccionados.

Suele utilizarse con las siguientes opciones:

-dX donde "X" es un caracter que hace de delimitador de campos. Si no se especifica, se toma por defecto el tabulador.

-fN donde "N" es el número de campo que se desea ver. Puede establecerse un rango: -fN-M, imprimirá desde el n-ésimo hasta el m-ésimo campo.

--complement

Imprime el complemento.

Ejemplos:

Supongamos que tenemos un archivo llamado columnas.txt con el siguiente formato:

```
Uno:Dos:Tres:Cuatro
Once:Doce:Trece:Catorce
Veintiuno:Veintidos:Veintitres:Veinticuatro
```

El comando

```
cut -d: -f2 columnas.txt
```

tomará como delimitador el carácter ":", y mostrará el segundo campo:

```
Dos
Doce
Veintidos
```

El comando
cut -d: -f2 --complement columnas.txt
mostrará todo el archivo excepto el segundo campo:
Uno:Tres:Cuatro
Once:Trece:Catorce
Veintiuno:Veintitres:Veinticuatro
El comando
cut -d: -f2-4 columnas.txt
mostrará del segundo al cuarto campo:
dos:tres:cuatro
Doce:Trece:Catorce
Veintidos:Veintitres:Veinticuatro
En cambio el comando
cut -d: -f2,4 columnas.txt
(con una coma en lugar de un guion entre el 2 y el 4) mostrará el segundo y el cuarto campo:
dos:cuatro
Once:Catorce
Veintidos:Veinticuatro

head

Muestra el comienzo de un archivo (por defecto, diez líneas).
Se suelen utilizar las siguientes opciones:
-cN, donde N es un número, imprimirá los primeros N bytes del archivo. Si después del número se agrega una k o una m, se mostrarán los primeros N kilobytes o megabytes respectivamente.
-nN Muestra las primeras N líneas.
Ejemplos:
head -c3k archivo.txt
Muestra los primeros 3kB del archivo.
head -n4 archivo.txt
muestra las primeras 4 líneas del archivo.

tail

Es similar a **head**, pero imprime las últimas líneas en lugar de las primeras.
Se utilizan también las opciones **-c** y **-n**, que funcionan igual que en **head**.
La opción **-f** permite "seguir" un archivo, para ver sus últimas líneas a medida que el archivo va cambiando. Suele ser útil para ver los cambios que se van haciendo en un *log* (archivo de registro).

sort

Muestra en forma ordenada la entrada que se le proporcione. Suele utilizarse con las siguientes opciones:
-f No distingue mayúsculas de minúsculas.
-k POS1 Ordena según la clave que empieza en POS1.
-n Para ordenar números.
-r Para ordenar al revés.
-t X donde X es el carácter que actúa como separador de campos.
-R orden aleatorio
-b ignora los espacios en blanco al comienzo de la línea.

Ejemplos:

Supongamos que tenemos un archivo llamado columnas.txt con el siguiente formato:

```
Uno:Dos:Tres:Cuatro:5
Once:Doce:Trece:Catorce:15
Veintiuno:Veintidos:Veintitres:Veinticuatro:25
```

El comando

```
sort -t : -k 4 columnas.txt
```

mostrará la salida ordenada por el cuarto campo:

```
Once:Doce:Trece:Catorce:15
Uno:Dos:Tres:Cuatro:5
Veintiuno:Veintidos:Veintitres:Veinticuatro:25
```

El comando **`sort -rt: -k4 columnas.txt`**

Mostrará la salida ordenada por el cuarto campo al revés:

```
Veintiuno:Veintidos:Veintitres:Veinticuatro:25
Uno:Dos:Tres:Cuatro:5
Once:Doce:Trece:Catorce:15
```

El comando **`sort -t: -k5 columnas.txt`**

Mostrará la salida ordenada alfabéticamente por el 5º campo: (15; 25; 5):

```
Once:Doce:Trece:Catorce:15
Veintiuno:Veintidos:Veintitres:Veinticuatro:25
Uno:Dos:Tres:Cuatro:5
```

En cambio, el comando **`sort -nt: -k5 columnas.txt`**

mostrará la salida ordenada numéricamente por el 5º campo: (5; 15; 25):

```
Uno:Dos:Tres:Cuatro:5
Once:Doce:Trece:Catorce:15
Veintiuno:Veintidos:Veintitres:Veinticuatro:25
```

Si se desea mostrar la salida de **`ls -l`** ordenada por el grupo al que pertenecen los archivos (4ª columna):

```
ls -l | sort -k4
```

tac

Este comando es, literalmente, el comando **`cat`** al revés. Muestra un archivo empezando por el último renglón y terminando por el primero.

tr

Se utiliza de la siguiente manera:

`tr [opciones] [cadena 1 [cadena2]]` (Los corchetes indican que el elemento es opcional).

"Traduce" (reemplaza) cada carácter de una cadena por el correspondiente carácter de otra cadena. Por lo tanto, ambas cadenas deben tener la misma cantidad de caracteres.

Este comando no tiene un argumento para especificarle un archivo. Si se quiere utilizar sobre un archivo, se debe utilizar el "entubamiento" (**`|`**) o la redirección (**`>`**).

Las cadenas pueden contener varios tipos de caracteres especiales, por ejemplo:

`a-z` Cualquier letra minúscula.

`\\` El carácter "barra invertida" (\ ).

Varios caracteres de control:

\a "Campana" - \b backspace - \n Salto de línea - \t Tabulador horizontal - \v Tabulador vertical

El comando tr suele utilizarse con las siguientes opciones:
-c Usa el complemento (todos los caracteres que *no* están en la cadena).
-d Borra los caracteres de la primer cadena.
-s Elimina los caracteres repetidos en la primer cadena.
Ejemplos:
Para convertir todas las letras de archivo1.txt a mayúsculas:

```
cat archivo1.txt | tr a-z A-Z
```

o bien:

```
cat archivo1 | tr '[:lower:]' '[:upper:]'
```

Una lista más completa de los parámetros usados en el último ejemplo es:
[:alnum:] todas las letras y los dígitos
[:alpha:] todas las letras
[:blank:] espacios en blanco
[:cntrl:] todos los caracteres de control
[:digit:] todos los dígitos
[:graph:] todos los caracteres imprimibles, sin incluir los espacios.
[:lower:] todas las letras minúsculas.
[:print:] todos los caracteres imprimibles, sin incluyendo los espacios.
[:punct:] todos los signos de puntuación
[:space:] todos los espacios horizontales o verticales
[:upper:] todas las letras mayúsculas
[:xdigit:] todos los dígitos hexadecimales

Para eliminar espacios en blanco repetidos en archivo1.txt:

```
cat archivo1.txt | tr -s '[:blank:]'
```

Para eliminar todos los caracteres no imprimibles de archivo1.txt, excepto el salto de línea:

```
cat archivo1.txt | tr -dc '[:print:]\n'
```

En todos estos ejemplos, los archivos quedan intactos. Los caracteres reemplazados solo se muestran por la salida estándar.

WC

Si bien ya lo habíamos utilizado ampliamente, no lo habíamos mencionado en los apuntes.
Este comando sirve para contar caracteres (**-c**), palabras (**-w**) y líneas (**-l**).
Ejemplos:

```
wc archivo.txt
```

Cuenta los caracteres, palabras y líneas de archivo.txt.

```
wc -c archivo.txt
```

Cuenta los caracteres de archivo.txt

```
ls -l | wc -l
```

Cuenta la cantidad de líneas que tiene la salida del comando ls -l

uniq

Es un comando que sirve para detectar líneas repetidas **adyacentes**. Por este motivo, suele ser utilizado con el comando **`sort`**.

Por defecto, el comando **`uniq archivo`** muestra todo el archivo, excepto que cuando hay líneas repetidas, se muestran una sola vez.

Sus opciones más comunes son las siguientes:

`-f N` donde N es un número, omite comparar los primeros N campos, siendo el delimitador de campos el espacio en blanco o tabulador.

`-s N` donde N es un número, omite comparar los primeros N caracteres.

`-c` muestra la cantidad de veces que se repite cada línea.

`-i` no distingue mayúsculas de minúsculas.

`-d` no muestra las líneas no repetidas.

`-u` no muestra las líneas repetidas (ni siquiera su primera aparición).

`-w N` donde N es un número, compara solamente los primeros N caracteres.

grep

Otro comando conocido a esta altura. Sirve para buscar una determinada expresión en un archivo (o en la entrada estándar). Su uso es:

`grep [opciones] expresión [archivos]`

Busca "expresión" en los archivos especificados (o en la entrada proporcionada). Por defecto, se muestran las líneas en las que aparece "expresión", y no se muestran las que no tienen coincidencia.

Opciones frecuentes:

`-c` Muestra solo la cantidad de líneas que coinciden, pero no el texto de las líneas.

`-h` Cuando se proporcionan múltiples archivos, el comando muestra un prefijo con su nombre. Si se desea omitir dicho prefijo, debe utilizarse la opción -h.

`-i` Para que no distinga mayúsculas de minúsculas (A=a)

`-n` Muestra el número de cada línea en donde se da la coincidencia.

`-v` Complemento. Muestra la líneas que *no* coinciden con "expresión".

`-E` Extensión. Permite el uso de expresiones regulares extendidas. **`grep -E`** equivale a **`egrep`**

sed

sed, *stream editor*, es un poderoso programa para filtrar texto. Suele utilizarse para automatizar tareas de edición repetitivas, o para procesar texto que llega mediante "entubamiento" (pipe). Su sintaxis es:

`sed [opciones] 'comando' [archivos]`

Invoca un comando de sed y lo aplica sobre los archivos.

`sed [opciones] -e 'comando1' [-e 'comando2' ...] [archivos]`

Invoca varios comandos de sed y los aplica sobre los archivos. El parámetro -e es obligatorio antes de cada comando.

`sed [opciones] -f script [archivos]`

Ejecuta los comandos que están en el archivo llamado script.

En cualquiera de los tres casos, conviene poner el comando entre comillas simples. Si los archivos no se especifican, sed opera sobre la entrada estándar (redirección o entubamiento).

`sed` opera sobre el texto utilizando direccionamiento y comandos. Cada línea de la entrada se procesa individualmente, sin relacionarla con las líneas adyacentes. Si se aplican varios comandos, se aplican todos en orden para la primera línea, luego se aplican todos a la segunda, etc.

Direccionamiento en sed

Las direcciones pueden ser:

Un número de línea (si se usan dos archivos y el primero tiene, por ejemplo, 10 líneas, la primer línea del segundo archivo será la línea 11). El símbolo **$** representa la última línea. Puede representarse un conjunto de líneas separando con coma (**inicio,fin**). Por ejemplo, el direccionamiento para todas las líneas sería **1,$**.

Una expresión regular delimitada por barras: **/expresionRegular/**

Un número de línea con un intervalo **n~s**, donde **n** es la línea de inicio, y **s** es el intervalo. Por ejemplo, si se quiere direccionar todas las líneas impares: **1~2** (empieza en la línea 1, y va de dos en dos).

Si no se especifica un direccionamiento, los comandos se aplican a todas las líneas de la entrada. Si un direccionamiento va seguido del símbolo **!**, se afectan todas las líneas que *no* coincidan con el direccionamiento.

Comandos sed

El *comando sed* sigue inmediatamente la especificación de direccionamiento (si está presente). Los comandos generalmente están compuestos por una sola letra o símbolo, a menos que tengan argumentos. Los comandos más utilizados son:

d Para borrar líneas

p imprime la línea si la sustitución tuvo éxito.

w archivo Imprime la línea en "archivo" cuando una sustitución tuvo lugar.

y "traduce" caracteres. Funciona parecido a tr (ver páginas anteriores).

s Para sustituir. Es un comando muy utilizado. La sintaxis es:

s/patrón/reemplazo/[banderas]

Las banderas pueden ser:

- **g** Reemplaza todas las apariciones de "patrón", no solo la primera.
- **un número** Si se especifica el número **n**, se reemplaza la n-ésima aparición de patrón.
- **I** No distingue mayúsculas de minúsculas
- **Nota:** estas banderas no funcionan de la misma manera en todas las implementaciones de sed

Ejemplos:

Borrar las líneas 5, 6 y 7 de archivo.txt

```
sed '5,7d' archivo.txt
```

Borrar las líneas de archivo.txt que comiencen con un numeral (#).

```
sed '/^#/d' archivo.txt
```

Cambiar todas las a por x, las b por y y las c por z.

```
sed y/abc/xyz/
```

Escribir una X en todas las líneas vacías de archivo.txt

```
sed 's/^$/X/g' archivo.txt
```

Eliminar todas las comillas de archivo.txt

```
sed 's/"//g' archivo.txt
```

Es importante destacar que, en todos los ejemplos anteriores, el texto modificado es enviado a la salida estándar, y archivo.txt queda intacto. Si se desea afectar el contenido del archivo, puede usarse la opción -i. Así, el último ejemplo quedaría:

```
sed –i 's/"//g' archivo.txt
```

Si se desea hacer un backup de archivo.txt:

```
sed –i.bak 's/"//g' archivo.txt
```

La versión original es **archivo.txt.bak**, y la versión modificada es **archivo.txt** (Puede elegirse cualquer sufijo, distinto de **.bak**, yo lo elegí arbitrariamente)

Expresiones regulares

Muchas aplicaciones y lenguajes utilizan expresiones regulares. Todas tienen una sintaxis similar, aunque no idéntica, ya que los intentos por especificar un único estándar comenzaron después de que las expresiones regulares fueran utilizadas en numerosas herramientas.

Las expresiones regulares se definen mediante unas cadenas de texto llamadas patrones. Estos patrones se componen de caracteres literales, y de metacaracteres (caracteres que representan otra cosa, como **^** y **$** para comienzo y fin de línea, o la lista de expresiones **[:clase:]** detallada más arriba).

Otros elementos de expresiones regulares:

. Un carácter cualquiera

\<ola\> Delimitador de palabra. Solo la palabra "ola" coincidirá con la búsqueda. Las palabras Hola, Sola, molar, etc, no coincidirán.

[abc] La a, la b, o la c (cualquiera de las tres).

[a-f] Cualquier letra minúscula entre la a y la f (a, b, c, d, e o f).

[^abc] [^a-f] Cualquier caracter que no sea ni la a, ni la b, ni la c (en el primer ejemplo); o cualquier carácter que no sea una letra entre la a y la f (en el segundo). (No confundir con el ^ que representa el comienzo de una línea).

Modificadores en expresiones regulares:

Con **grep**	Con **egrep** o **grep -E**	Descripción
*****	*****	Cero o más veces la expresión regular precedente.
\?	**?**	Cero o una vez la expresión regular precedente.
\+	**+**	Una o más veces la expresión regular precedente.
\{n\}	**{n}**	donde n es un número, n veces la expresión regular precedente.
\{n,\}	**{n,}**	n o más veces la expresión regular precedente.
\{n,m\}	**{n,m}**	entre n y m veces la expresión regular precedente.
|	**|**	"O". La expresión precedente o bien la expresión siguiente. **a|b** --> la "a" o la "b"
\(expr\)	**(expr)**	Para agrupar.

Ejemplos:

Buscar en archivo1.txt la palabra Urquiza o urquiza:

```
grep '[uU]rquiza' archivo1.txt
```

Mostrar las líneas de archivo1.txt que tienen 3 dígitos juntos:

```
grep '[0-9][0-9][0-9]' archivo1.txt
```

o bien:

```
egrep '[0-9]{3}' archivo1.txt
```

Mostrar las líneas de archivo1.txt que no comiencen con una letra mayúscula:

```
grep '^[^A-Z]' archivo1.txt
```

Mostrar todas las líneas de archivo1.txt que tengan al menos un carácter:

```
grep '.' archivo1.txt
```

Mostrar todas las líneas de archivo1.txt que tengan el caracter . (punto). Es necesario utilizar el carácter de escape \, para distinguirlo del metacarácter:

```
grep '\.' archivo1.txt
```

Mostrar todas las líneas que consten únicamente de tres letras a o tres letras b:

```
egrep '^[ab]{3}$' archivo1.txt
```

Mostrar cualquier número de 3; 4 o 5 cifras en archivo1.txt

```
grep '\<[0-9]\{3,5\}\>' archivo1.txt
```