Edición de textos

Sumario

Vi y Vim

Vim es un potente editor, descendiente del clásico vi, que integra capacidades avanzadas de edición de archivos de texto. Es un editor que requiere cierto tiempo de adaptación, ya que no es tan intuitivo como otros editores como nano. Vim opera en dos modos de trabajo, el **modo de edición** y el **modo normal**. En el primero podremos editar los contenidos del fichero, mientras que el segundo se utiliza para ejecutar comandos relacionados con la gestión de aspectos no relacionados con la edición del texto, como por ejemplo borrados, sustituciones, búsquedas, etc.

Para acceder al modo de edición pulsamos la tecla **i**. Para acceder al modo normal pulsamos la tecla **Esc**.

Comandos básicos en el modo normal para gestionar el guardado, salida y edición de otro archivo

"Listado de comandos de vi":http://www.cs.colostate.edu/helpdocs/vi.html

- **:q** Salir del editor sin guardar (quit)
- **:q!** Salir del editor sin guardar ni pedir confirmación (quit ya!)
- **:wq!** Salir del editor guardando sin pedir confirmación (write and quit ya!)
- **:w f2.txt** Guardar en un fichero llamado f2.txt y seguir
- **:e f1.txt** Cierra el fichero actual y abre f1.txt (edit f1.txt)

Desplazamientos, inserciones y sustituciones

- **w**: salta al inicio de las palabras
- **b**: salta palabra a palabra hacia atrás
- **e**: salto al fin de las palabras
- ^: salto al primer carácter no blanco al inicio de línea
- **0**: salta al inicio de línea
- $: salto al fin de línea
- **x**: elimina el carácter actual
- **X**: elimina el carácter anterior
- **o**: inserta línea debajo de la línea actual
- **O**: inserta línea antes de la línea actual
- **i**: inserta texto antes de posición actual
- **I**: insertar al principio de línea

- **p**: pega el contenido del búfer después del cursor
- **P**: pega el contenido del búfer antes del cursor
- **ce**: sustituye la palabra actual
- **C:** sustituye toda la línea actual
- **a**: inserción de texto a continuación del cursor
- **A:** inserción de texto al final de la línea

Uso de las flechas del teclado

Aunque se recomienda el uso de las teclas **h, j, k, l** para desplazamientos por el texto en modo normal, al principio es más intuitivo hacerlo con las flechas del teclado. Para habilitar esta capacidad, ya que en algunas terminales estas teclas son interpretadas como caracteres de escape y escriben los códigos en el editor vim, añadiendo símbolos extraños al texto, editamos el archivo **.vimrc** dentro del directorio home del usuario, y añadimos las líneas:

```
nnoremap <Left> h
nnoremap <Right> l
nnoremap <Up> k
nnoremap <Down> j
inoremap <left> h
inoremap <right> l
inoremap <up> k
inoremap <down> j
```

Comandos para copiar, cortar y pegar texto

Copiado

- **yw**: Copia la palabra en la posición del cursor, para copiar toda la palabra éste debe estar posicionado al principio de la misma
- **yy**: Yank (copiado) de la línea actual
- **y$**: copia de la posición actual del cursor al fin de línea
- **y0**: copia de la posición actual del cursor al principio de línea
- **5yy**: copia 5 líneas

Pegado

- **p**: pega después del cursor
- **P**: pega antes del cursor

En el modo normal (Esc) pulsamos **v** (visual), entraremos en modo seleccionar. Nos desplazamos por el cursor para abarcar el texto seleccionado y pulsamos **c** para cortar el texto o **y**, para copiarlo. Para pegar el texto copiado o cortado pulsamos, en la posición destino, **p** Otro modo más sencillo para cortar es utilizar **dd**, al utilizar este comando se corta la línea actual, o varios si ponemos dn (n númerode líneas a cortar), pero se copia en el registro por defecto, al estilo portapapeles, luego usamos **p** y pegamos el contenido del registro en la posición indicada

Reemplazo

- **r**: Reemplaza el carácter bajo el cursor
- **R**: Reemplaza a partir del carácter bajo el cursor
- **cw**: Reemplaza la palabra actual
- **C**: Reemplaza desde el carácter del cursor hasta el final de línea

Borrado

- **dd**: Borra toda la línea actual
- **dNd**: Borra las N líneas a continuación
- **dw**: Borra la palabra bajo el cursor
- **D**: Borra desde el cusor hasta el final de línea
- **d0**: Borra desde la posición del cursor hasta el principio de línea

Comandos para operaciones con texto

- **dd** Suprimir línea actual al buffer (p para pegar) (delete)
- **u** Deshacer el último cambio en el fichero (undo)

- **CTRL+R** Rehacer el último cambio en el fichero (redo)
- **guu** Convertir a minusculas la línea actual (lowercase)
- **gUU** Convertir a mayúsculas la línea actual (UPPERCASE)
- **:num** Posicionarse en la línea num del fichero
- **gg** Posicionarse al principio del fichero
- **G** Posicionarse al final del fichero
- **ga** Muestra código ASCII, hex y octal del caracter actual

Búsqueda y sustitución con vim

En modo normal

- **Buscar texto**:
  - **:/palabra** (pulsaremos **n** para ir a la siguiente ocurrencia o **N** para la anterior)
- **Sustituir texto**:
  - **:%s/texto1/texto2** sustituye texto1 por texto2 en la línea actual
  - **:%s/texto1/texto2/g** con la terminación /g se sustituyen todas las ocurrencias de texto1 por texto2, sino se indica se sustituye la primera ocurrencia
  - **:%s/texto1/texto2/gi** igual que la anterior pero case insensitive
  - **5,12s/texto1/texto2** sustituye en el rango de líneas 5 a 12 texto1 por texto2
  - **:'<,'>s/texto1/texto2** en modo visual sustituye en la selección texto1 por texto2 (el '<,'> se incluye automáticamente tras la selección visual)
  - **:g/^baz/s/texto1/texto2/g** cambia en todas las líneas que empiecen por baz texto1 por texto2
  - **:.,$s/texto1/texto2/g** cambia todas las ocurrencias de texto1 por texto2 desde la línea actual hasta el final del archivo
  - **:.,+2s/texto1/texto2/g** cambia todas las ocurrencias de texto1 por texto2 en la línea actual y las 2 siguientes

Autocompletar

Puede usarse **autocomplección**:

- Utilizando palabras en el propio archivo en modo inserción con **ctrl+n**, ocurrencia siguiente, o **ctrl+p**, ocurrencia posterior. Aparecerá una lista con las sugerencias encontradas
- Utilizando rutas en el sistema de archivos, usando **ctrl+x ctrl+f**
- **Autocompletar utilizando un archivo externo a modo de plantilla:**

Puede hacerse también abriendo en la misma sesión de vim otro archivo del que tomar las palabras para autocompletar a modo de diccionario, por ejemplo para insertar directivas tomadas de un archivo de configuración de pruebas. Para ello usamos **:split archivo_plantilla**, esto abrirá el archivo en modo split, dividiendo el área de trabajo. Saltamos con **ctrl+w** al área anterior y ya podemos, en modo inserción, utilizar el autocompletado con **ctrl+n** y **ctrl+p** tomando las palabras del archivo incluído.

Opciones de configuración

En el archivo **vimrc**, suele estar en /**vimrc**, se definen las opciones de configuración de vim. Algunas

- **:set ts=3** Fija los tabulados a 3 espacios
- **:set sw=3** Fija los indentados a 3 espacios
- **:set number/nonumber** Activa/desactiva el numerado en los ficheros
- **:set backup/nobackup** Activa/desactiva la copia de seguridad automática
- **:set directory=dir** Fija la carpeta donde se harán las copias
- **:syntax on/off** Activa/desactiva el resaltado de sintaxis
- **:color esquema** Cambia color del vim (evening, darkblue, desert, elflord, koehler, morning...)
- **:set cindent** Activa indentado automático
- **:set mouse=a/mouse=** Activa/desactiva el uso del ratón
- **:set paste/nopaste** Activa/desactiva el modo pegar texto literalmente
- **:spell** Activa el corrector ortográfico
- **:setlocal spell spelllang=es** Activa el idioma español del corrector ortográfico
- **:set spellfile=~/.vimdic** Fija diccionario de palabras desconocidas

Ejecutar comandos externos

Es posible que mientras estamos editando un fichero, necesitemos ejecutar un comando (por ejemplo, un ls para ver los archivos), esto se puede hacer escribiendo en el modo normal **:!comando**, en nuestro ejemplo: **:!ls**. También se puede hacer una pausa en la edición escribiendo **:shell** para realizar alguna operación y cuando la terminemos, escribir **exit** y volver al editor. Incluso los comandos **:make** y **:cc** se pueden utilizar para ahorrarnos el estar saliendo del editor para hacer makes o ver el último error que nos dió.

Editar varios ficheros a la vez

Con **:split fichero** (división horizontal) o **:vplit fichero** (división vertical) pueden editarse varios archivos de texto a la vez. Para conmutar de unas a otras áreas de edición se utilizaría **CTRL+W** y desplazamiento con cursor.

Pueden utilizarse también varias pestañas, al estilo Firefox, para ello escribimos en modo normal **:tabnew fichero**. Para movernos con ellas utilizamos **:tabn**, y pulsando > o con el ratón movernos por ellas.

Otros comandos útiles

- **CTRL+N**: Autocompletar texto
- **=G** Indenta automáticamente todas las líneas de un fichero
- **{}** Detecta donde está la llave mal cerrada del párrafo actual
- :g/^\s**$/d Elimina las líneas en blanco de un fichero**

Algunos trucos

- **ELIMINAR LÍNEAS EN BLANCO DE UN ARCHIVO**: Desde el modo :, ejecutamos **g/^$/d**. La opción g sirve para indicar una expresión de regular de búsqueda, en este caso **^$** (líneas vacías) y a continuación indica una acción a realizar con esas líneas, en este caso la opción **d** (borrar)
- **ELIMINAR LÍNEAS DE COMENTARIO**: Igual que en el ejemplo anterior, la expresión sería **g/^*/d**

Listado completo de comandos vim

VIM

Cursor movement

h - move left
j - move down
k - move up
l - move right
w - jump by start of words (punctuation considered words)
W - jump by words (spaces separate words)
e - jump to end of words (punctuation considered words)
E - jump to end of words (no punctuation)
b - jump backward by words (punctuation considered words)
B - jump backward by words (no punctuation)
0 - (zero) start of line
^ - first non-blank character of line
$ - end of line
G - Go To command (prefix with number - 5G goes to line 5)
Note: Prefix a cursor movement command with a number to repeat it. For example, 4j moves down 4 lines.

Insert Mode - Inserting/Appending text

i - start insert mode at cursor
I - insert at the beginning of the line
a - append after the cursor
A - append at the end of the line
o - open (append) blank line below current line
(no need to press return)
O - open blank line above current line
ea - append at end of word
Esc - exit insert mode

Editing

r - replace a single character (does not use insert mode)
J - join line below to the current one
cc - change (replace) an entire line
cw - change (replace) to the end of word
c$ - change (replace) to the end of line
s - delete character at cursor and subsitute text
S - delete line at cursor and substitute text (same as cc)
xp - transpose two letters (delete and paste, technically)
u - undo
. - repeat last command

Marking text (visual mode)

v - start visual mode, mark lines, then do command (such as y-yank)
V - start Linewise visual mode
o - move to other end of marked area
Ctrl+v - start visual block mode
O - move to Other corner of block
aw - mark a word
ab - a () block (with braces)
aB - a {} block (with brackets)
ib - inner () block
iB - inner {} block
Esc - exit visual mode

Visual commands

> - shift right
< - shift left
y - yank (copy) marked text
d - delete marked text
~ - switch case

Cut and Paste

yy - yank (copy) a line
2yy - yank 2 lines
yw - yank word
y$ - yank to end of line
p - put (paste) the clipboard after cursor
P - put (paste) before cursor
dd - delete (cut) a line
dw - delete (cut) the current word
x - delete (cut) current character

Exiting

:w - write (save) the file, but don't exit
:wq - write (save) and quit
:q - quit (fails if anything has changed)
:q! - quit and throw away changes

Search/Replace

/pattern - search for pattern
?pattern - search backward for patt
n - repeat search in same direction
N - repeat search in opposite directi
:%s/old/new/g - replace all old wi
throughout file
:%s/old/new/gc - replace all old w
throughout file with confirmations

Working with multip

:e filename - Edit a file in a new buf
:bnext (or :bn) - go to next buffer
:bprev (of :bp) - go to previous buf
:bd - delete a buffer (close a file)
:sp filename - Open a file in a new b
window
ctrl+ws - Split windows
ctrl+ww - switch between windows
ctrl+wq - Quit a window
ctrl+wv - Split windows vertically

INVOCACIÓN VI.

$vi????????Editar un texto sin nombre $vi archivo?????-Editar un archivo (nuevo o no) $vi archivo1 archivo2??Editar lista de archivos $vi +n archivo????-Editar archivo en la línea n $vi +/txt archivo???-Editar archivo en la 1a línea donde aparece txt

MOVIMIENTOS DEL CURSOR.

Arriba?-k Abajo??j Derecha?h Izquierda-l

0???Inicio de línea $???Fin de línea w???Word: Avanzar palabra b???Back: Retroceder palabra e???End: Al final de palabra H???Home: Esquina sup. izq. de la ventana L???Last: Esquina inf. izq. de la ventana crtl+u?-Window up: Subir ventana crtl+d?-Window down: Bajar ventana crlt+b?-Page back: Retroceder página crtl+f?-Page forward: Avanzar página nG???Go: Salta a la línea n. 1G???A la primera línea $G???A la última línea fcar??Buscar en la línea el carácter car (hacia delante) Fcar??Buscar en la línea el carácter car (hacia atrás) INSERTAR TEXTO.

i?Insertar (delante del cursor) I?Insertar al principio de la línea a?Añadir (detrás del cursor) A?Añadir al final de la línea o?Insertar una línea debajo de la actual O?Insertar una línea encima de la actual BORRAR TEXTO.

x?Borrar caracter actual X?Borrar caracter anterior dd?Borrar línea actual D?Borrar hasta final de línea dw?Borrar palabra CAMBIAR TEXTO.

rcar?Reemplazar el caracter actual por car R??Reemplazar texto desde la posición del cursor s??Substituir el caracter actual por texto a insertar S??Substituir la línea actual C??Cambiar hasta el final de la línea cw?-Cambiar palabra J??Unir a la línea actual la siguiente COPIAR Y PEGAR.

yy?-Copiar en el buffer la línea actual nyy?Copiar en el buffer n líneas desde la actual p??Pega el buffer detrás del cursor P??Pega el buffer delante del cursor BUSCAR Y SUBSTITUIR.

%??????-Busca el caracter delimitador ( ) [ ] { } que balancea el actual (Dentro de un entorno salta al

delimitador inicial) /ExpReg????-Busca hacia delante la expresión regular ExpReg ?ExpReg????-Busca hacia atrás la expresión regular ExpReg n??????-Repite la última búsqueda N??????-Repite la última búsqueda en el sentido contrario

s/txt/txt2???Substituye el texto txt por txt2 la primera vez que aparece en la línea
s/txt/txt2 /g??Substituye todas las apariciones de txt por txt2 en la línea
m,n s/txt/txt2 /g?Substituye en el rango de líneas [m,n]

REPETIR Y DESHACER.

.?Repetir último comando de actualización (Borrado/Inserción/Cambio) u?Deshacer último comando de actualización U?Deshacer todos los cambios en la línea actual COMANDOS DEL SHELL.

sh????Invoca un nuevo shell. Al salir continua la edición
!CmdShell?-Ejecuta un comando del sistema operativo
r!CmdShell?Ejecuta un comando del S.O. e inserta su salida en la posición del cursor
!!????Repite el último comando ejecutado en un shell

OPERACIONES CON ARCHIVOS.

w?-Graba las modificaciones efectuadas en el archivo
w?-archivo Escribe el texto actual en archivo (Sólo si no existía)
q?-Salir (si no hay cambios)
q!?Salir (sin grabar)
wq?Guardar cambios y salir
x ?-Guardar cambios y salir

ZZ?-Guardar cambios y salir ESTADISTICAS DE ARCHIVO.

=??Muestra el número total de líneas del archivo
.=??Muestra el número de línea actual

crtl+G?Muestra el nombre del archivo, línea actual, número total de líneas y porcentaje recorrido del archivo. OPCIONES DE ENTORNO.

set opción??Activa la opción de vi correspondiente

set noopción?Desactiva la opción de vi correspondiente

all??-Muestra todas las opciones y sus valores number?-Muestra numeración de líneas list??Muestra caracteres de control ic???Ignora mayúsculas/minúsculas en las búsquedas

sed

sed, stream editor, es una utilidad de edición de texto en modo línea de comandos que permite ejecutar comandos sobre las líneas de uno o varios archivos de entrada línea a línea. El modo de operación es tal que se procesa cada una de las líneas, delimitadas por caracteres de retorno de carro, y se le aplican los comandos especificados en la sintaxis de llamada a sed. A modo de resumen, sed opera del siguiente modo:

- Lectura de una línea desde el flujo de entrada (las líneas están delimitada por un carácter de salto de línea)
- La línea es procesada en función de los comandos leídos
- Muestra (o no) del resultado en la salida estándar (pantalla)
- Continúa con la línea siguiente

La **sintaxis** es la siguiente:

sed [-opciones] [comando] [<fichero(s)>]

detallando las **opciones**:

sed [-n [-e comando] [-f script] [-i[.extension]] [l [corte]] rsu] [<comando>] [<fichero(s)>]

- **-n**, --quiet, --silent

  ```
  Solicitud implícita para no mostrar el estado de la memoria principal (buffer). En un script la notación se hará de esta manera "
  ```

- **-e script**, --expresión=script

  ```
  Permite encadenar varios comandos
  ```

- **-f fichero-script**, **--file=fichero-script**

  ```
  Lectura de comandos desde el fichero indicado
  ```

- **-i[SUFIJO]**, --in-place[=SUFIJO]

  ```
  Edita archivos en el lugar. También da la posibilidad de hacer una copia de respaldo añadiendo la extensión (-i.BAK)
  ```

- **-r**, --regexp-extended

  ```
  Utiliza expresiones regulares extendidas (ERE)
  ```

- **-s**, --separate

  ```
  Si varios ficheros son ingresados en la entrada, los procesa uno a uno en vez que como uno solo
  ```

- **-u**, --unbuffered

  ```
  Carga cantidades mínimas de datos desde los ficheros de entrada y libera los almacenamientos temporales de salida con mayor frecu
  ```

- **--help**

  ```
  Muestra esta ayuda y termina
  ```

- **--version**

  ```
  Muestra información acerca de la versión del programa y termina.
  ```

Algunos **comandos**:

- **p**: Imprimir

Ejemplo:

```
sed 's/unix/linux/p' file.txt
```

El comando p muestra cada una de las líneas sustituidas. Como sed escribe en la salida estándar cada una de las líneas procesadas, las líneas a las que se ha efectuado la sustitución aparecerán 2 veces en la salida

- **s**: Sustitución

Ejemplo:

```
sed 's/original/nuevo/g' archivo.txt
```

Sustituye todas las ocurrencias del texto "original" por "nuevo" en archivo.txt. Notar que las expresiones van precedidas por el carácter /

- **d**: Borrado

Ejemplo:

```
sed '1,5d' archivo.txt
```

Elimina las líneas 1 a 5 de archivo.txt

```
sed '1,/Jack/d' poem.txt
```

Elimina desde la línea 1 hasta la aparición del texto "Jack"

- **i**: Inserción

Ejemplos:

```
sed '4 i Si tú me dices ven' archivo.txt
```

Inserta antes de la línea 4 de archivo.txt (para insertarlo antes utilizaríamos la opción **a** en lugar de **i**) el texto "Si tú me dices ven"

```
sed '/Sysadmin/a Lo dejo todo' archivo.txt
```

Añade a continuación de las líneas en las que aparezca el texto "Sysadmin", el texto "Lo dejo todo"

- **c**: Cambio

```
sed '/unix/ c "nuevo texto"' file.txt
```

Cambia las líneas que contengan el texto unix por "nuevo texto" en el archivo file.txt

```
sed -r '1,7s/[a-z]+/nuevotexto/' file.txt
```

Sustituye, comando s, **entre las líneas 1 a 7** del archivo file.txt aquellas líneas que verifiquen el patrón de expresión regular, para eso habilitamos las expresiones regulares con **-r**. Busca expresiones alfanuméricas en minúsculas, [a-z]+, y sustituye las ocurrencias por "nuevotexto". Como no se utiliza la opción **-i**, no se modifica el contenido del archivo original Comando para obtener un listado de los directorios del PATH:

```
ls -la echo `echo $PATH | sed 's/:/ /g'`
```

Direcciones sed como se ha visto reconoce direcciones a la hora de especificar los comandos de procesamiento. Se pueden especificar de varios modos

- **numero**: indicando el número de la línea
- **inicio~paso**: indica las líneas n-ésimas (según el valor de paso), a partir de la línea inicial (según inicio)
- /**expr_regular**/: indicando las líneas que verifican la expresión regular indicada

Por tanto el uso de expresiones regulares POSIX es común en la definición de comando sed

AWK

Awk es un lenguaje de programación utilizado para procesar información en modo texto. Trabaja de modo similar a sed en el sentido que procesa archivos de texto línea por línea, aplicando patrones de búsqueda y ejecutando acciones definidas en el comando.

La sintaxis principal de awk es:

awk '/patron_busqueda/ {acciones;}' archivo

Donde

- **patron_busqueda** (opcional): es una expresión regular que se buscará en cada una de las líneas del archivo. Sino se especifica se procesarán todas las líneas
- **{acciones;}**: conjunto de acciones, separadas por ;, que representan los comandos a ejecutar en cada línea procesada en el archivo que verifique la condición establecida en el patrón de búsqueda. Por defecto, sino se especifica acción, se imprimirá por defecto la línea entera

Patrón de búsqueda

En esta expresión aplicaremos expresiones regulares para determinar la coincidencia textual

- Se utilizarán los operadores **==, <, >, >=, >=, !=** para expresar comparación directa o literal en la condición
- Se utilizará el operador ~ para indicar comparación de campo mediante el uso de una expresión regular
- En la condición establecida por el patrón de búsqueda pueden utilizarse los operadores **&&** (y lógico) y || (o lógico)

Claúsulas BEGIN y END

Si se usan estas cláusulas la sintaxis es

awk 'BEGIN {acciones;} /patron_busqueda/ {acciones;} END {acciones;}' archivo

- La claúsula **BEGIN** se ejecuta antes de procesar cualquier línea del archivo
- La claúsula **END** se ejecuta tras procesar todas las líneas del archivo

Su uso más habitual es inicializar variables o ejecutar una acción global de inicio y fin del comando awk

Uso de scripts externos

Para scripts o expresiones complejas con awk es posible utilizar archivos externos en la que definir los comandos y patrones. Para ejecutar awk tomando como entrada un archivo de expresiones awk utilizamos la opción **-f**

```
awk -f archivo.awk
```

Ejemplos de Uso

- **Imprimir todas las líneas de un archivo**

```
awk /etc/passwd
```

- **Imprimir las líneas de un archivo según un patrón textual**

```
awk '/root/' /etc/passwd
```

En este caso buscaría en las líneas de /etc/passwd aquella que contenga el texto root

- **Seleccionar las líneas vacías de un archivo**

```
awk '/^$/' archivo
```

- **Contar las líneas de un archivo**

```
awk '{x=x+1};END {print x}' /etc/passwd
```

El texto dentro del archivo se procesa línea por línea, **RECORDS** para awk, y dentro de cada línea se identificarán los campos, **FIELDS** para awk.

Utilizamos los siguientes identificadores para hacer referencia a las partes de cada RECORD

- **$0**: la línea completa
- **$1, $2, $3...**: FIELDS según su posición, $1, primer FIELD, campo o columna de la línea, por defecto se utiliza como carácter de separación el espacio, o conjunto de espacios. Este valor puede redefinirse en la variable **FS**
- **$NF**: Número de FIELDS de la línea. Muy útil para hacer referencia al último FIELD de la línea

Otras variables

- **NR**: Número de RECORDs (líneas)
- **RS**: Separador de RECORD, por defecto el salto de línea
- **FS**: Separador de FIELD de entrada
- **OFS**: Separador de FIELD de salida
- **CR**: RECORD actual
- **CF**: FIELD actual

Más ejemplos:

- **Imprimir el primer campo de un archivo**

```
awk '{print $1}' /etc/passwd
```

- **Imprimir el primer campo cuando ese campo verifique una condición de expresión regular**

```
awk '$1 ~ /^r/ {print $1}' /etc/passwd
```

- **Imprimir el primer y quinto campo de un archivo de las líneas que empiecen por r**

```
awk -F: '/^r/ {print $1" "$5}' /etc/passwd
```

En este caso entre el $1 y el $5 se ha introducido un " " para insertar un espacio en blanco entre los campos, necesario pues awk no procesa los espacios en la expresión del print. La opción **-F** permite redefinir el separador de campo, en este caso a ":" que es el carácter que delimita los campos en cada línea dentro del archivo /etc/passwd. El comando anterior sería equivalente a

```
awk 'BEGIN {FS=":"}
/^r/ {print $1" "$5}' /etc/passwd
```

- **Contar las líneas vacías de un archivo**

```
awk 'BEGIN {x=0}
/^$/ {x=x+1}
END {print "Hay " x " lineas vacias en el archivo"}' /etc/passwd
```

Prestar atención a la posición de los *y a las comillas del mensaje de texto del END* **Suma de dos números**

```
echo "7 8" | awk '{print $1+$2}'
```

Raíz cuadrada de un número

```
echo 10 | awk '{print sqrt($1)}'
```

Seno de un número

```
echo "7" | awk '{print sin($1)}'
```

Ejecutar una suma

```
echo "1 2 3 4 5" | awk 'BEGIN {suma=0;RS=" "};{suma=suma+$CF};END {print "la suma es " suma}'
```

Cálculo de las primeras potencias de 2

```
echo "1 2 3 4 5 6 7 8 9 10" | awk 'BEGIN {RS=" "};{print 2^$CF}'
```

Producto de varios números

```
echo -e "1\n3\n5\n8" | awk 'BEGIN {res=1};{res=res'''$CF};END {print res}'
```

Referencias

Vim tips and tricks

https://www.cs.oberlin.edu/~kuperman/help/vim/

Search replace

http://vim.wikia.com/wiki/Search_and_replace

Usos del comando sed

http://www.grymoire.com/Unix/sed.html

Manual sed

http://lowfatlinux.com/linux-sed-manual.html

Ejemplos sed:

http://www.folkstalk.com/2012/01/sed-command-in-unix-examples.html

AWK

https://www.ibm.com/developerworks/library/l-awk1/

Expresiones regulares

https://manuais.iessanclemente.net/index.php/Exmpresions_Regulares_en_Linux http://www.tutorialspoint.com/unix/unix-regular-expressions.htm

Volver

JavierFP 16:31 08 ene 2019 (CET)