GNU/Linux para programadores:

Examinar y manipular contenido de archivos por línea de comandos

En ediciones anteriores comentábamos lo importante que es que un programador tenga la capacidad de moverse libremente por línea de comandos y aprendimos a movernos a través del sistema de archivos. Ahora, nos enfocaremos en las diversas formas que tenemos para examinar internamente, documentos de texto de cualquier tipo y manipular su contenido.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y **eXtreme Programming**. Miembro de la **Free Software Foundation**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: @eugeniabahit

Al igual que la vez que hablamos sobre el manejo y manipulación de archivos y directorios, iremos viendo desde las opciones más comunes a las más complejas, comenzando por aquellas que quizás te resulten más familiares. Es el caso de `cat`: el comando que utilizamos para "leer" un archivo.

A decir verdad, `cat`, concatena dos o más archivos mostrando el resultado en pantalla:

```
eugenia@cococha-gnucita:~/borrador/uno$ echo "soy el archivo a" > a
eugenia@cococha-gnucita:~/borrador/uno$ echo "soy el archivo hola" > hola
```

```
eugenia@cococha-gnucita:~/borrador/uno$ cat a hola
soy el archivo a
soy el archivo hola
eugenia@cococha-gnucita:~/borrador/uno$
```

Sin embargo, el mismo comando también se utiliza solo para **ver en pantalla el contenido de un archivo**:

```
eugenia@cococha-gnucita:~/borrador/uno$ cat .htaccess
RewriteEngine On
RewriteRule !(^static|favicon) app_engine.php
AddType image/x-icon .ico
RewriteRule ^favicon favicon.ico [NC,L]
eugenia@cococha-gnucita:~/borrador/uno$
```

El comando `cat` es especialmente útil para visualizar *scripts* o cualquier otro archivo de texto plano, exceptuando aquello dinámicos y/o de gran tamaño como archivos de *logs*.

Para visualizar archivos de gran tamaño, disponemos de tres opciones: a) **visualizar las últimas líneas de un archivo** lo cual es especialmente útil para visualizar el último error o datos de acceso en, por ejemplo, los *logs* de Apache:

```
eugenia@mynewdream:~/ruta/a/logs$ tail -n 2 access.log
189.188.26.118 - - [09/Apr/2013:22:06:13 -0300] "GET /static/pdf/material-sin-
personalizar-python.pdf HTTP/1.1" 206 33104
"http://www.cursosdeprogramacionadistancia.com/static/pdf/material-sin-
personalizar-python.pdf" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.31 (KHTML,
like Gecko) Chrome/26.0.1410.43 Safari/537.31"
189.188.26.118 - - [09/Apr/2013:22:06:16 -0300] "GET /static/pdf/material-sin-
personalizar-python.pdf HTTP/1.1" 206 914949
"http://www.cursosdeprogramacionadistancia.com/static/pdf/material-sin-
personalizar-python.pdf" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.31 (KHTML,
like Gecko) Chrome/26.0.1410.43 Safari/537.31"
eugenia@mynewdream:~/ruta/a/logs$
```

En la muestra anterior, `-n 2` indica que se desean ver 2 líneas. Se puede indicar la cantidad deseada.

La opción b) es **visualizar las últimas líneas de un archivo en tiempo real**, lo que es especialmente útil para auditar logs y saber todo lo que sucede en tiempo real. Esto se logra pasando `-F` como argumento en vez de `-n`:

```
tail -F logs/error.log
```

Es importante aclarar que `-F` es sinónimo de `-f` (en minúsculas).

Y luego, la tercera opción u opción c), es **leer las primeras líneas de un archivo** lo cual es útil en líneas generales para visualizar, por ejemplo, archivos binarios:

```
eugenia@mynewdream:~$ head -n3 php.pdf
%PDF-1.5
%
3 0 obj
eugenia@mynewdream:~$ head -n3 agile.pdf
%PDF-1.4
%äüöß
2 0 obj
```

o también, cualquier archivo de texto convencional:

```
eugenia@cococha-gnucita:~/borrador/ramtesting/fibo$ head -n2 foop
#!/usr/bin/php
<?php
eugenia@cococha-gnucita:~/borrador/ramtesting/fibo$ head -n2 foopl
#!/usr/bin/perl

eugenia@cococha-gnucita:~/borrador/ramtesting/fibo$
```

Cuando un archivo es de gran tamaño y sin embargo necesitamos verlo completo, podemos utilizar `less` o `more`: **`cat ARCHIVO | less`** (luego avanzamos con Av Pág o flecha abajo) **`cat ARCHIVO | more`** (avanzamos con la tecla intro)

Muchas veces, **buscar texto dentro de un archivo** nos será de mucha ayuda.

```
eugenia@cococha-gnucita:~/borrador$ grep -e "key" template.py
    def render_regex(self, stack, key):
        dicc = dict(key=key)
        regex = re.compile('<!--%(key)s-->(.|\n){1,}<!--%(key)s-->' % dicc)
        render = render.replace('<!--%s-->' % key, '')
eugenia@cococha-gnucita:~/borrador$
```

Para lo cual podremos utilizar la siguiente sintaxis:

```
grep -e "EXPRESION" DONDE

Busca todos los archivos que comiencen por <?php cuya extensión sea .php
eugenia@cococha-gnucita:~/borrador$ grep -e "^<?php" *.php
ejercicios_sql_290113.php:<?php
ejercicios SQL.php:<?php
lala.php:<?php
settings.php:<?php
eugenia@cococha-gnucita:~/borrador$
```

El comando grep también es sumamente útil cuando se desea auditar *logs* en tiempo real. Pero en este caso la sintaxis será diferente:

```
COMANDO | grep -e "EXPRESION"
```

Dónde `COMANDO`, podría ser cualquiera (no solo `tail`). Pero veamos un ejemplo con `tail` para auditoría en tiempo real:

```
eugenia@mynewdream:~/ruta/a/logs$ tail -F access.log | grep -e "GET\ \/analisis"
190.194.58.46 - - [09/Apr/2013:22:38:37 -0300] "GET /analisis HTTP/1.1" 200 58535
"http://www.cursosdeprogramacionadistancia.com/analisis" "Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/537.31"
190.194.58.46 - - [09/Apr/2013:22:38:58 -0300] "GET /analisis HTTP/1.1" 200 58535
"http://www.cursosdeprogramacionadistancia.com/curso-poo" "Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/537.31"
^C
eugenia@mynewdream:~/ruta/a/logs$
```

Dentro de la manipulación interna de un archivo, la forma más básica es **escribir texto sin utilizar un editor (>)**:

```
eugenia@cococha-gnucita:~/borrador$ echo "lalala" > documento
eugenia@cococha-gnucita:~/borrador$ cat documento
lalala
eugenia@cococha-gnucita:~/borrador$ echo "Hola Mundo" > documento
eugenia@cococha-gnucita:~/borrador$ cat documento
Hola Mundo
eugenia@cococha-gnucita:~/borrador$
```

Como se puede ver, este comando sobre-escribe el contenido del archivo. Si no se desea sobre-escribir el contenido, es necesario **agregar texto al final del archivo (>>)**:

```
eugenia@cococha-gnucita:~/borrador$ echo "Ahora no lo sobre-escribo" >> documento
eugenia@cococha-gnucita:~/borrador$ cat documento
Hola Mundo
Ahora no lo sobre-escribo
eugenia@cococha-gnucita:~/borrador$
```