PROGRAMACIÓN INTEGRATIVA

PRIMERA SESIÓN PRÁCTICA: CONTENIDO DE AMPLICACIÓN

En esta parte de la sesión práctica se harán varios comentarios acerca de las opciones más básicas de los comandos vistos en la primera sesión: `sed`, `grep` y `find`, .

sed

`sed` es una herramienta bastante potente para realizar transformaciones básicas de texto. Hemos visto anteriormente cómo hacer substituciones básicas. La forma más simple es:

```
sed --expresion='s/A/B/' <file>

sed -e 's/A/B/' <file>

sed 's/A/B/' <file>
```

Las tres son equivalentes: sustituir A por B en el fichero `file`. Se utiliza la opción `-e` o `--expression` cuando se combinan comandos:

```
sed -e 's/A/B/' -e 's/A/B/' <file>
```

Por defecto, `sed` ante un fichero o un *stream* irá línea por línea realizando las substituciones pertinentes. Sin embargo, sólo lo hará una vez por línea. Imaginemos que tenemos el siguiente fichero:

```
Foo bar foo bar
bar foo bar
foo bar foo bar
bar foo foo
```

Si aplicamos `sed -e 's/foo/FOO/' <file_sed>`, el resultado será:

```
Foo bar FOO bar
bar FOO bar
FOO bar foo bar
bar FOO foo
```

Como vemos, en la tercera línea el segundo `foo` no se ha sustituido. En la primera línea, dado que `sed` es *case sensitive*, tampoco ha sustituido el primer `Foo`. Por ello, para que sed actúe de manera global y de forma *insensitive*, debemos colocar `gi` al final de la expresión:

```
sed 's/foo/FOO/gi' <file_sed>
FOO bar FOO bar
bar FOO bar
```

```
FOO bar FOO bar
bar FOO FOO
```

A las expresiones también se le puede aplicar un rango de actuación. Si quisiésemos que sólo se substituyesen en las filas 2 y 3, por ejemplo, lo colocaríamos al principio de la expresión:

```
sed '2,3s/foo/FOO/gi' <file_sed>
Foo bar foo bar
bar FOO bar
FOO bar FOO bar
bar foo foo
```

Si se especifica un rango seguido de `!`, se seleccionan las líneas fuera de ese rango, p. ej: 2,3! en el caso anterior, escogería las líneas 1 y 4.

PREGUNTA

¿Cómo eliminarías en ese fichero el segundo `foo` de la primera línea y ambos `foo` de la cuarta con sólo una expresión?

Más información y otros enlaces de interés:

- `https://linux.die.net/man/1/sed`
- `http://www.grymoire.com/Unix/Sed.html#uh-28`

---

grep

Como sabemos, utiliza expresiones regulares para filtrar la entrada recibida. Es un comando muy útil para buscar líneas en los ficheros, por ejemplo. La sintaxis básica es:

```
grep ``string" file1 file2 ... # diferentes ficheros
grep ``string" file* # ficheros que empiecen por 'file'
grep ``string" -r '.' # de forma recursiva
```

Es mencionable que `grep` no sólo busca en ficheros regulares, si no que también detecta apariciones en ficheros binarios. Algunas opciones interesantes de esta herramienta (entre muchas otras):

- `-c`: cuenta el número de líneas en las que aparece el patrón.
- `-o`: mostrar sólo la parte de la línea en la que se encuentra el patrón.
- `-l`: mostrar los ficheros en los que se encuentra el patrón.
- `-n`: mostrar el número de línea.
- `-w`: buscar sólo las palabras completas que coinciden con el patrón.
- `-i`: *case insensitive*.
- `-v`: invertir el sentido del *matching*.

Cogiendo el fichero que creamos para la sección anterior, por ejemplo, mostrar las líneas en las que aparece el patrón `bar`, seguido de cualquier cadena y de nuevo `bar`, y sólo mostrar esa parte de la línea:

```
grep -on "bar.*bar" file_sed
1:bar foo bar
2:bar foo bar
3:bar foo bar
```

PREGUNTA

Para el fichero que se muestra a continuación:

```
Foo bar nofoo bar
bar foo bar
nofoo nobar nofoo
foo bar foo bar
nofoo nobar nofoo
bar foo foo
```

¿Cómo sacarías las líneas completas y sus números de línea de aquellas que no contienen la palabra `foo`?

Más información y otros enlaces de interés:

- `https://linux.die.net/man/1/grep`
- `https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux`

`find`

Como hemos visto, `find` lista de forma recursiva y exhaustiva los ficheros y directorios de la ruta actual. Este comando dispone de muchas opciones y tests. Podemos destacar:

- `-name` y `-iname`: se utiliza para filtrar por nombre de fichero/directorio. En el segundo caso sin importar mayúsculas.
- `-mtime` n: para filtrar por fecha de modificación ($n * 24$ horas).
- `-empty`: para listar sólo los ficheros vacíos.
- `-not`: para negar las demás opciones, es decir, listar aquellos elementos que no cumplan las demás opciones.
- `-exec` *command*: para ejecutar un comando sobre aquellos ficheros/directorios listados.
- `-user`, `-uid`, `-group`, etc.: filtrado por usuario, número de identificación de usuario, grupo, etc.

Con el contenido del siguiente directorio, vamos a realizar diferentes pruebas:

```
[user@host:/usr]$ ls -la
total 2,0M
-rw-rw-r-- 1 usr  usr   564 Set 19 16:06  70e4f.torrent
-rw-rw-r-- 1 usr  usr  457K Set 19 16:19  DNI.pdf
-rw-rw-r-- 1 usr  usr  3,1K Set 11 13:14  file
-rw-rw-r-- 1 usr  usr  755K Set 19 12:52  lee_micro09_talk.ppt
-rw-rw-r-- 1 root root 3,1K Set 11 13:53  new_file
```

```
-rw-rw-r-- 1 root root 3,1K Set 11 13:52  new_file.bak
-rw-rw-r-- 1 usr  usr     0 Set 11 13:47  other
drwxrwxr-x 5 usr  usr  4,0K Set 21 12:45  pi
-rw-rw-r-- 1 usr  usr   41K Set 17 13:47  README
-rw-rw-r-- 1 usr  usr   42K Set 17 13:47 'README (1)'
-rw-rw-r-- 1 usr  usr  5,9K Set 18 12:32  ResizedPhotos.0.zip
lrwxrwxrwx 1 usr  usr    24 Set 11 11:04  symb -> /home/Imaxes
-rw-rw-r-- 1 usr  usr  650K Set 19 12:42  TMA_Metrics.xlsx
```

Mostrar aquellos **ficheros** en **este directorio** que sólo hayan sido **modificados hace tres días**:

```
[user@host:/usr]$ find . -type f -mtime 3
./README (1)
./README
./ResizedPhotos.0.zip
```

Mostrar aquellos **ficheros vacíos** en **este directorio**:

```
[user@host:/usr]$ find . -type f -empty
./other
```

Listar aquellos **ficheros** del **root** en **este directorio** que terminen en **.bak** y eliminarlos utilizando `-exec`:

```
[user@host:/usr]$ find . -type f -user root -iname ``*.bak" -exec rm '{}' \;
```

Como se ve en el ejemplo, se utiliza `'{}'` para pasar la salida del `find` como parámetro a `rm`. Además, el comando que ejecuta `-exec` termina con un punto y coma (;) pero escapado (`\;`) para evitar interpretaciones del *shell*.

Otra posible forma de escribir lo anterior es utilizando la opción `-delete`:

```
[user@host:/usr]$ find . -type f -user root -iname ``*.bak" -delete
```

PREGUNTA

¿Qué diferencia hay entre los siguientes tres comandos?

```
find . -type f -user root -iname ``*.bak" -exec grep *.bak '{}' \; -print

find . -type f -user root -iname ``*.bak" | xargs grep *.bak

find . -type f -user root -iname ``*.bak" | grep *.bak
```

Más información y otros enlaces de interés:

- `https://linux.die.net/man/1/find`
- `https://www.linode.com/docs/tools-reference/tools/find-files-in-linux-using-the-command-line/`