Taller de Química Computacional Aplicada: Sesión 1

Rony J. Letona

12 de mayo de 2017

1. Ejercicios con la línea de comando

1.1. Archivos y Directorios

Navegar dentro del sistema de archivos de Linux puede hacerse por medio de una interfaz gráfica. Estas son las que ofrecen ventanas, íconos, botones, barras de búsqueda, etc. Sin embargo, habrán veces en las que ver el contenido de una carpeta no será práctico por medio de una interfaz gráfica. Además, en ella no se puede crear rutinas de manera inmediata. Por eso se recomienda tener una noción de cómo navegar dentro del sistema de archivos mediante la línea de comando. A continuación haremos algunos ejercicios para comprender mejor cómo hacer esto.

1.1.1. Directorio de Trabajo

Lo primero que debemos saber antes de movernos hacia otro directorio es en qué directorio nos encontramos ahorita. Para ello vamos a utilizar un comando que nos revelará esa información. Abre la terminal (línea de comando), después del signo `~ $` escribe `pwd` y presiona Enter. Cuál fuel el resultado? Anótalo y recuérdalo.

1.1.2. Navegando entre carpetas

Ahora intentaremos cambiar de lugar dentro del árbol de directorios. Ingresa el siguiente comando `cd /` y presiona Enter. Después de eso, vuelve a revisar cuál es el directorio en el que te encuentras. Hay alguna diferencia? Si sí, cuál y por qué crees que la hubo?

1.1.3. Listando Contenidos

Finalmente, ingresa el comando `ls` y presiona Enter. Qué puedes ver allí? Reconoces alguno de los archivos que allí se te muestran? Toma nota de ellos. Ahora ingresa `cd` solamente y presiona Enter. Repite el procedimiento con `ls` y anota lo que ves. Qué crees que hace el comando `ls` ?

1.2. Creando Cosas

Todos los archivos en un ordenador son archivos de texto. En algunos, el texto es incomprensible para nosotros; es código que solo el ordenador puede leer. Mientras que en otros, este es diseñado para ser comprendido por nosotros. Para crear archivos de texto, generalmente usamos procesadores de texto como Word, WordPad en Microsoft Windows. Sin embargo, para crear un simple archivo de texto, a veces hemos utilizado Notepad o Block de Notas. En Linux también existen procesadores de texto con una interfaz gráfica, como Gedit o Kate. Sin embargo, estos todavía son elaborados. Para esta sección crearemos y borraremos directorios, y haremos una serie de ejercicios entre los que crearemos archivos de texto en la línea de comando y los borraremos después.

1.2.1. Directorios

Para trabajar mejor, vamos a crear un nuevo directorio en donde guardar nuestros ejercicios y pruebas. En la terminal que teníamos del ejercicio anterior, vamos a ingresar un nuevo comando: `mkdir Playground`

Para revisar qué es lo que ha pasado, vamos a listar el contenido de el directorio en el que nos hallamos y revisar qué cambios ha habido desde que listamos los contenidos la vez pasada. Notas el nuevo directorio?

Ahora procederemos a entrar a ese directorio de la siguiente manera `cd Playground` y luego crearemos un directorio de prueba llamado *Test*. Cómo lo harías? Inténtalo!

1.2.2. Archivos de Texto

En el caso de los archivos de texto, cuando los creamos en la línea de comando, no seguimos el mismo patrón que en una interfaz gráfica. En la última, abrimos el editor, creamos el archivo y finalmente lo guardamos. Si bien esto se puede hacer en algunos editores de la línea de comando, es recomendable nombrar el archivo desde el inicio y no hasta el final. Hagamos una prueba. Ingresa el siguiente comando: `nano mi_archivo.txt` y analiza lo que ves.

Ahora escribe algún mensaje dentro de tu nuevo archivo y ciérralo. Lee muy bien cada instrucción que se te de en el proceso. Al final lista el contenido de la carpeta y nota cómo es que ahora aparece tu nuevo archivo.

1.2.3. Eliminar

Hasta ahora todo va muy bien, sin embargo, estas han sido pruebas y debemos dejar nuestro directorio limpio antes de continuar con el taller. Para ello, necesitamos eliminar tanto el directorio *Test* como nuestro recién-creado archivo. A diferencia de simplemente seleccionar a cada uno y eliminarlos, en la línea de comando generalmente se utilizan dos comandos diferentes: uno para archivos (`rm`) y otro para carpetas (`rmdir`). Intenta eliminar ambos.

Nota: Al eliminar un directorio, este tiene que hallarse vacío. De no ser así, el comando no dejará que lo elimines y debes usar otra manera un poco más peligrosa. Si deseas saberla, pregunta por ella.

1.2.4. Copiar y Pegar

Una práctica que nos encanta al trabajar editando texto en ordenadores es el de copiar, cortar y pegar. Esto quizá se deba a que no existe un equivalente en el mundo real, excepto quizá por las fotocopias. Sin embargo, estas no son tan fieles como una copia digital: copia 100 % idéntica. Para realizar este tipo de operaciones con nuestros archivos, vamos a copiar un archivo del directorio de documentos a nuestro directorio, y luego vamos a cortar otro archivo del directorio de documentos y lo trasladaremos a nuestro directorio de trabajo.

Comenzando, copiaremos un archivo con el siguiente comando: `cp ../TQCA/Data/lorem_ipsum.txt ~/Playground/`
Ahora, debemos tomar en cuenta un par de aspectos importantes. El primero son los dos puntos antes del directorio de documentos. Qué crees que hagan? Y luego está el hecho de que primero pusimos la dirección de todo el archivo, luego solo la ubicación donde lo íbamos a colocar. Anota eso, es importante.

Finalmente, cortaremos y pegaremos un archivo. Como ya nos dimos cuenta antes, en estos casos no se copia y pega, sino que se transfiere una copia de manera directa. En este caso es igual, solo que a transferir un archivo en Linux se le denomina *mover*. Por ello, el comando utilizado es `mv` Elimina el archivo *lorem_ipsum.txt* en tu directorio de trabajo e ingresa este comando en una nueva línea: `mv ../TQCA/Data/lorem_ipsum.txt ~/Playground/lorem_ipsum.txt`
El comando es casi igual al de copiado, solo con una diferencia importante. Notas cuál es?

1.3. Permisos y Propiedad

En muchos casos nos vamos a topar con que nuestro ordenador no nos deja crear un documento nuevo o una carpeta. Tampoco nos deja eliminar algunos archivos o modificarlos. Detente y piensa a qué crees que se debe esto? Nuestro ordenador protege algunos documentos y carpetas. Muchas veces nos toparemos con que el *administrador* tiene que permitirnos algo, porque si no tenemos sus privilegios, no podemos modificar nada.

En nuestro caso en particular, el sistema operativo en el que nos encontramos protege algunas cosas. Sin embargo, vamos a aprender que podemos accesar a ellas y permitir su modificación o negarla según lo que creas conveniente. Primero debemos de tener claro que, en Linux, cada documento tiene un propietario. Además de esto, los documentos pueden ser vistos por 3 tipos de usuarios o categorías. Veamos esto directamente.

1.3.1. Qué [no] se puede hacer?

Entramos a nuestra carpeta mediante `cd Playground` y luego escribimos `ls -l` . Si ponemos atención, lo que hicimos fue pedir el listado de documentos dentro de la carpeta, pero esta vez obtuvimos una gran cantidad de información. Nos interesa un resultado que se ve algo así `-rw-r--r--` . Vamos a descomponer esto en sus partes:
La secuencia comienza con un caracter que nos dice con qué estamos tratando.

- `-` es un documento o archivo regular.
- `d` es un directorio o una carpeta.
- `l` es un enlace simbólico[1].

Luego nos topamos con 3 triadas de caracteres. Estas nos dicen lo que se puede hacer con cada uno por parte de cada usuario o categoría. Y qué se puede hacer con él?

- `r` nos dice que el documento puede ser leído.
- `w` nos dice que el documento puede ser editado o que puede escribirse en él.
- `x` nos dice que el documento o archivo puede ejecutarse como un programa.
- `-` nos dice que alguna de las 3 acciones anteriores no puede ser realizada.

Ahora que ya entendimos las acciones que se pueden hacer con cada uno, nos debemos preguntar: por qué vienen en triadas? Aquí entra lo de los 3 usuarios o categorías. Un archivo puede ser leído, escrito o ejecutado por: el propietario, el grupo de usuarios al que pertenece y una categoría especial llamada *otros*. Esta última representa a cualquier usuario (aunque no sea el propietario del documento o alguien en su grupo de trabajo).

Entonces, en nuestro caso, de haber obtenido algo como `-rw-r--r--` nos damos cuenta de que tratamos con un documento o archivo regular. Este podemos leerlo y podemos escribir en él. Sin embargo, no podemos ejecutarlo! Por otra parte, los miembros de nuestro grupo de trabajo y los otros usuarios pueden leer el documento, pero no pueden hacer nada más con él. Esta es la manera en que podemos ir identificando qué permisos tiene cada documento o carpeta con la que nos topamos. Qúe pasaría si cambiaras de directorio y revisas los permisos de otros documentos?

Más interesante que ver los permisos de los documentos, es cambiarlos. Para esto debemos entender que tanto el propietario, el grupo o la categoría otros pueden representarse con una `u` , `g` u `o` respectivamente. Ahora vamos a cambiar los permisos de un documento. Para agregar permisos utilizamos un `+` , mientras que para quitarlos utilizamos un `-` . Pero el comando que realmente cambia los permisos es `chmod` . Entonces, para agregarle permisos de lectura y escritura a algún documento *mi_documento.txt*, escribimos `chmod u+rw mi_documento.txt`
Ahora, si deseamos quitarle permisos de ejecución a un programa pequeño *mi_programa.sh*, escribimos
`chmod u-x mi_programa.sh` . Si nos damos cuenta, lo que hicimos fue cambiar permisos para el propietario nada más. Si desearamos cambiar permisos para el grupo de usuarios actual u *otros*, podemos reemplazar la `u` por una `g` o una `o` respectivamente.

1.3.2. De quién [no] es esto?

Ya entendimos cómo se quita y pone permisos, a quién darle los permisos y cómo averiguar cuáles son. Lo que nos queda ahora es saber cómo cambiar el propietario de un documento o el grupo al que este pertenece. Veamos cómo hacer esto. Si deseamos saber qué usuario es el propietario de un documento, lo que debemos de hacer es igual a lo que vimos en la sección anterior: `ls -l` . Después de cada una de las representaciones de permisos, vienen 2 nombres. En nuestro caso probablemente serán `mint` y `mint` , pero porque tanto el usuario que estamos usando como el grupo en el que este se encuentra se llaman `mint` . En muchas ocasiones nos veremos frente a casos en donde el usuario y el grupo son distintos (e.g. nuestro usuario es nuestro nombre y el grupo es el grupo de trabajo de la empresa o equipo de investigación en el que nos hallamos).

Si ya sabemos a quién pertenece cada documento y en qué grupo está, lo que nos queda es saber cómo cambiar esto. Al igual que en el caso anterior, existe un comando que hace el cambio de propietario. Este es el comando

[1] Esto lo veremos más adelante

`chown` . Para cambiar un documento *mi_documento.txt* del usuario actual al usuario *root*, por ejemplo, lo que debemos escribir es `chown root mi_documento.txt` [2]. Si en vez de cambiar el usuario al que pertenece el documento deseáramos cambiar el grupo en el que se encuentra, escribimos `chown :root mi_documento.txt` . Finalmente, si deseamos cambiar ambas cosas a la vez (asumiendo que previamente cambiamos ambas en un documento que habíamos elegido), escribimos `chown mint:mint mi_documento.txt` . Qué entendimos de esto? Qué indican los dos puntos a la hora de cambiar propietarios y grupos? Comenta con tu compañero de al lado.

Ya que entendimos que existen diferentes tipos de permisos, de propietarios y de grupos para cada documento dentro de nuestro ordenador, pensemos un momento y comentemos con nuestros compañeros sobre las implicaciones que esto tiene. Por qué tanto permiso y tanta distinción? Qué se gana al restringir lo que se cada persona puede hacer con distintos documentos o programas? Qué impacto tiene esto en la seguridad y eficiencia de nuestro ordenador?

1.4. Vínculos

Cuando utilizamos un ordenador por primera vez, aprendemos el concepto de un *Acceso Directo* o un *Hipervínculo*. Esto solo resulta ser un nombre o un ícono que apunta a otro documento o programa. Cuando lo presionamos o hacemos doble click sobre él, nuestro ordenador va a la dirección contenida en este nombre y abre o ejecuta un documento. Esto es muy común en Windows. En Linux no se acostumbra tanto a trabajar con esto, aunque sí existe un equivalente que vamos a ver en esta sección.

Un vínculo o *link* puede ser representado por un documento, ícono o nombre dentro de nuestro sistema operativo. En Linux existen dos tipos de vínculos: los duros y los suaves o simbólicos. La diferencia es bastante sencilla, pero vamos a ver un ejemplo para entender cómo se hace cada uno.

1.4.1. Vínculos Duros

Este es el tipo de vínculo más sencillo de hacer. Para esto vamos a escoger (o crear) un documento *mi_documento.txt* y vamos a crear un vínculo de él. Para ello escribimos `ln mi_documento.txt nuevo_documento.txt` . Si ponemos atención, dentro de nuestra carpeta donde estaba el documento, acaba de aparecer otro llamado *nuevo_documento.txt*. Es más! Si revisamos el contenido de este, nos daremos cuenta que es idéntico al de nuestro documento original. Podríamos hacer modificaciones al contenido de uno, que las veríamos reflejadas en el otro. Podríamos ver cómo se ven estos documentos al escribir `ls -l` . Podríamos también cambiar el nombre de *mi_documento.txt* a otra cosa y revisar el contenido de ambos otra vez notando que este sigue siendo el mismo. Sin embargo, *nuevo_documento.txt* parece ser un documento; no hay nada que evidencie que solo sea un nombre o un ícono. Qué estará pasando? Comenta con tu compañero de al lado e intenta explicarlo. En el ínterin, vamos a pasar a los vínculos suaves o simbólicos.

1.4.2. Vínculos Simbólicos

La diferencia entre el comando para crear vínculos duros y simbólicos es muy sencilla. Para crear un vínculo simbólico de un documento *mi_documento_2.txt* escribimos `ls -s mi_documento_2.txt nuevo_documento_2.txt` . La diferencia fue sencilla: `-s` . Esto es porque debemos especificar que el vínculo es simbólico. Ahora vamos a realizar todas las pruebas que realizamos en el caso anterior. Primero revisaremos el contenido de ambos documentos para ver si es el mismo. Ya? Ahora pasemos a cambiar el contenido en uno y ver si ese cambio es evidente en los dos. Hasta aquí todo bien. Qué pasará si vemos la representación en el listado con `ls -l` ? Finalmente vemos un cambio! Claramente vemos que nuestro nuevo documento tiene color diferente, comienza su sistema de permisos con `l` y apunta (con una flecha) hacia el documento original. Ahora cambiaremos el nombre de *mi_documento_2.txt* y volvemos a revisar el contenido de los dos. Problema! El documento parece no existir. Esto lo vemos más claro si volvemos a escribir `ls -l` . Parece ser que nuestro vínculo no existe. Qué habrá pasado? Por qué es que esto ya no funciona? Discútelo con tu compañero de al lado e intenta ampliar tu explicación anterior sobre vínculos.

1.4.3. Explicación

Los documentos en nuestro ordenador son solo información guardada en un disco duro. Sin embargo, esta no tiene un orden en particular! Cómo sabe nuestro ordenador dónde hallar las cosas, entonces? De la misma manera en

[2] Si tu ordenador te dice que esta operación no es permitida, no entres en pánico. Pregunta por la manera de arreglar esto; no es nada complicada.

que nosotros hallamos la casa de un amigo la primera vez que vamos: utilizando direcciones. El disco duro contiene una tabla en la que guarda todos los nombres de los documentos en nuestro ordenador y sus direcciones en el disco duro. Entonces, al referirnos al nombre de un documento para abrirlo, nuestro ordenador busca en la tabla del disco duro, adquiere la dirección y, colocándose en la posicón del disco indicada por la dirección, lee el contenido y nos muestra el documento *abierto*.

Y qué tiene que ver esto con los vínculos? Que ahora que sabemos que el nombre de un documento y su contenido son dos cosas independientes, podemos entender mejor qué hace cada vínculo. Un vínculo duro es solo otro nombre dentro de la tabla de direcciones del disco duro, pero apuntando a la misma dirección de un documento existente; la del documento del que deseábamos un vínculo. Un vínculo simbólico es un nombre dentro de la tabla de direcciones en el disco duro, pero apuntando a otro nombre en la misma tabla. Por eso al cambiar el nombre de un documento, su vínculo simbólico deja de funcionar (el nombre al que se refería ya no existe) mientras que un vínculo duro no (el contenido del documento no se ha movido, así que la dirección sigue siendo la misma).

1.5. Tubos y Filtros

Una de las tareas más difíciles en el mundo actual es el saber qué información tomar en cuenta y cómo separarla de todos los datos que extraemos de experimentos, de la red y de muchas fuentes de información. Esta tarea comienza con simples filtros y pequeñas rutinas que nos ordenan los resultados de alguna manera. Así, la información no solo se vuelve más comprensible, sino que también se puede notar patrones, tendencias y otras cosas. Es por ello que ahora comenzaremos con otro tipo de operaciones sobre los archivos: aprenderemos a extraer información y a realizar más de un comando a la vez con la información que tenemos.

1.5.1. Contando Pedazos

Una tarea sencilla que podemos usar para extraer algo de información de un archivo es contar la cantidad de líneas, palabras o caracteres en él. Para esto existe un comando que nos muestra estos 3 resultados de una vez, o solo uno de los 3 si así lo deseamos. Para una pequeña prueba, comenzaremos con un conteo general: `wc ../TQCA/Data/lorem_ipsum.txt` Esto debió haberte dado como respuesta los 3 resultados de los que hablábamos.

Ahora, para contar solo las palabras, intentaremos algo diferente: `wc -w ../TQCA/Data/lorem_ipsum.txt` Notaste el argumento que agregamos? Intenta descubrir cómo contar solo líneas o solo caracteres.

Finalmente, si quisieramos ver cuántas palabras hay en todos los archivos de texto contenidos en la carpeta de documentos, podemos hacer lo siguiente: `wc -w ../TQCA/Data/*.txt` Aquí tenemos algo muy interesante que notar. En vez de usar un nombre para denotar un archivo en particular, utilizamos un `*` . Este nos permite referirnos a *todos* los archivos a la vez. Terminar con la extensión `.txt` filtra los resultados dejando solo a aquellos archivos con *esa* extensión en particular. Intenta repetir el ejercicio con la cantidad de líneas de cada archivo.

1.5.2. Tubos y Ordenando Listas

Y de qué nos sirve ahora tener todos los resulatdos allí? Pues ... es cierto. Esto no dice mucho. Pero qué pasaría si pudieramos trabajar más con esos resultados? Pues esa es la idea de los tubos! Tubo se le llama al comando que nos permite usar la información de salida de un comando, como información de entrada del siguiente. En la línea de comando de Linux, esto se hace mediante este caracter: `|` Hagamos un ejercicio.

Vamos a comenzar donde lo dejamos la vez pasada: con una lista de archivos y el conteo de la cantidad de palabras en cada uno. Ahora vamos a intentar ordenar los resultados. Como siempre, se sigue la convención de ir en el orden alfabético. Intentemos, pues, ordenar los resultados. Para eso ingresa lo siguiente en tu línea de comando: `wc -w ../TQCA/Data/*.txt | sort`

Creo que inmediatamente notaste los cambios. Agregamos el tubo (en inglés *pipe*) y agregamos un comando para ordenar nuestros resultados: `sort` Al fin tenemos todo ordenado, o no? Lo único que no está bien con el resultado es el total. El total, siendo un número mayor debería de ir al final. Para eso quizá sea necesario agregar un último argumento. Ingresa: `wc -w ../TQCA/Data/*.txt | sort -n` Ahora obtuvimos el resultado deseado; solo había que pedirle al comando de ordenar, que lo hiciera de manera numérica.

1.5.3. Hallando cosas

Al trabajar con muchos archivos, uno de los problemas con los que nos encontramos seguido es querer aislar solo algunos de ellos. Son bastantes los casos en los que un programa produce varios archivos de salida: resultados, documentación de la corrida, errores, etc. A nosotros quizá solo nos interese el de los resultados. Haremos un ejercicio para demostrar cómo se hace esto de manera sencilla. Ingresa en tu línea de comando lo siguiente: `find ../TQCA/Data/*ipsum*` y observa el resultado. Piensa y discute un momento sobre qué otro comando que ya conoces podría generar el mismo resultado.

Hasta ahora hemos trabajado con archivos solamente, no con su contenido. A lo más que habíamos llegado es a escribir dentro de un archivo con `nano` . Pero qué pasa si queremos ver lo que hay dentro de un archivo sin editarlo? Qué pasa si deseamos buscar cosas dentro del contenido? Para eso vamos a utilizar dos nuevos comandos. El primero nos permitirá ver el contenido de un archivo de texto sin abrirlo. Hagamos una prueba. Ingresa `cat ../TQCA/Data/lorem_ipsum.txt` y observa qué pasa.

Como te habrás dado cuenta, se muestra el contenido de todo el archivo. Esto puede ser lo que buscamos, o puede que no, ya que a veces, es solo el principio o el final del archivo lo que necesitamos. Para eso podemos usar `head -5` si quisieramos las primeras 5 líneas de un archivo, por ejemplo. O en el caso de querer las últimas 3, podemos ingresar `tail -3` . Por supuesto, los números los podemos cambiar, pero eso dependerá de lo que querramos en ese momento.

Ahora que ya sabemos cómo mostrar el texto de un archivo de formar total y parcial, vamos a buscar algo dentro de él. Para ello vamos utilizar un nuevo comando: `grep` . Este comando busca por expresiones regulares dentro de un texto y devuelve las líneas en el texto que contengan esa expresión. Una *expresión regular* puede ser una palabra o algunas fórmulas que permiten hallar patrones dentro del texto. En este caso vamos a proceder buscando un patrón sencillo dentro de un archivo con mucha información. Ingresa esto en la terminal y comenta sobre el resultado `cat ../TQCA/Data/data.csv | grep 199` . Hallamos entonces a todos los premios nobel en química de la década de los 90. Sin embargo, obtenemos también un resultado que no nos interesa: la última línea. Para eso, podemos contar cuántas líneas hay y así seleccionar solo las que deseamos. Antes de darte la manera de hacer esto, piensa un momento en cómo llevar a cabo esta tarea con lo que ya sabes.

Para resolver el problema anterior, lo que hacemos es agregar otro *pipe* al comando que ya teníamos, contar las líneas y con `head` seleccionar solo las que deseamos. En otras palabras, hacemos esto:
`cat ../TQCA/Data/data.csv | grep 199 | wc -l`
lo cual debe de darte como resultado `19` . La única línea que deseamos eliminar es la última, por lo que seleccionamos solo 18 líneas:
`cat ../TQCA/Data/data.csv | grep 199 | head -18`
De esa forma tenemos al listado de todos los premios nobel en química de la década de los 90s.

1.5.4. Filtrando Resultados

Los resultados obtenidos del último ejercicio ya son algo que nos provee información rápida sobre el contenido de un archivo. No obstante, si desearamos solo el año, el nombre y el apellido de cada premio nobel de química, tenemos que hacer un último arreglo. Como notamos de los resultados, el documento que utilizamos usa comas para dividir los campos. Y si ponemos atención, son las divisiones 1, 5 y 6 las que nos interesan para obtener la información antes mencionada. Veamos cómo hacer esto. En la terminal ingresa:
`cat ../TQCA/Data/data.csv | grep 199 | head -18 | cut -d , -f 1,5,6`
Observa los resultados y comenta sobre qué crees que hace cada parte del nuevo comando y argumentos que se utilizaron ahora.

Finalmente, tener resultados así en la línea de comando puede ser útil a veces, pero en el caso de haber extraído información importante, quizá es deseable almacenar esto en otro archivo. Ahora, en vez de copiar y pegar los resultados obtenidos del ejercicio anterior, vamos a alargar nuestro comando un poco más. Esta vez vamos a agregar un pequeño signo más: `>` . Este nos permitirá guardar los resultados obtenidos en un archivo de la siguiente forma:
`cat ../TQCA/Data/data.csv | grep 199 | head -18 | cut -d , -f 1,5,6 > mi_archivo.txt`
Después de ejecutar eso, como no obtuviste un resultado, deseo que veas cuál es el contenido de tu directorio de

trabajo y que lo comentes con los demás.

1.6. Ciclos

Perfecto, ahora ya sabemos cómo trabajar con un archivo. El asunto es que muchas veces, en especial cuando muestreamos, vamos a tener muchos datos en diferentes archivos. Cada archivo merecerá nuestra atención, puesto que los datos en ellos nos permitirán realizar una investigación o un estudio.

En esta sección vamos a ver cómo trabajar de manera repetitiva. Esto nos permitirá trabajar con varios archivos: uno a la vez, pero sin tener que escribir código para cada uno de ellos sino para todos de un solo. Para eso vamos a aprender sobre ciclos.

1.6.1. for ... in ...

El ciclo más práctico suele ser el ciclo `for` . Realmente, como casi todas las cosas dentro de un ordenador están enumeradas se les puede enumerar, es muy fácil utilizar este tipo de ciclos para resolver cualquier problema de repetición de operaciones. Vamos a ver un ejemplo sencillo de esto. Para eso vamos a utilizar una variable *enu* que nos servirá para enumerar. La sintaxis del comando será así:

```
~$ for enum in 1 2 3 4 5 6 7 8 9 0
> do
> echo $enum
> done
```

Esta vez hemos incluido varias cosas. Así que debemos ir una por una. `for` es el comando inicial que declara el comienzo de toda la instrucción. `in` denota que nuestra variable para enumerar va a tomar los valores que se hallan *en* la lista siguiente. `do` anuncia el comienzo de las instrucciones que van a repetirse. Y finalmente, `done` indica el final de estas instrucciones. Todo esto es necesario en *todo* comando cíclico `for` . Como podemos ver, no solo se trata de una palabra, sino que son varias ordenadas de una forma particular.

Ahora nos queda explicar qué es `echo` y por qué es que nuestra variable `enum` ahora comienza con un `$` `echo` es un comando que nos sirve para mostrar el valor de una variable en la terminal. Nuestra variable `enum` por otra parte, requiere del signo `$` para poder usar su valor. Si no agregamos ese signo antes de una variable, la terminal no nos dejará usar el valor que tiene almacenado esa variable.

1.7. Scripts

Para terminar con nuestra breve introducción a la línea de comando, vamos a intentar combinar todo lo que hemos aprendido y hacer algo interesante con ello. Supongamos que estamos trabajando en una investigación. Entre nuestros documentos hay ciertos archivos con datos de un estudio de aguas (falso) en el lago de Atitlán. Estos datos están en forma de tablas que contienen el pH, la temperatura de la muestra y la profundad a la que se tomó para 5 diferentes puntos de muestreo. Un ejemplo es el archivo *Agua_Ati_20140224.csv*

```
Punto,pH,Temp / °C,Prof / m
San Antonio,7.6724475943,23.61061926,10.4545873217
Santiago,7.2130122874,21.0422111279,10.677484815
San Pedro,7.7370670531,21.3379880926,9.4660710143
Santa Maria,7.046375392,20.4230312561,9.3680636119
Panajachel,7.7367713079,22.9446999496,10.160015882
```

La investigadora principal nos envía un correo electrónico solicitándonos las medias del pH, temperatura y profundidad para cada uno de los archivos. Adjunto nos envía instrucciones de que utilicemos una herramienta que puede calcular las medias de cada uno de los parámetros medidos, pero solo en **un** archivo. Esa herramienta está dentro de nuestros archivos y se llama *med.sh* Finalmente nos dice que esos datos le urgen para dentro de los próximos 15 minutos.

Hacer el trabajo a mano (un archivo a la vez) es fácil si la cantidad de archivos es baja, pero no sabemos si se trata de 3 archivos o de 3000. La idea entonces es automatizar todo el proceso. Desarrollaremos entonces un *script* que lleve a cabo esta tarea por nosotros y nos devuelva los resultados sin preocuparnos nosotros por la cantidad de archivos. Un script no es nada más que una serie de comandos, como los que hemos estado aprendiendo, escritos dentro de un archivo de texto para ser ejecutados secuencial- y automáticamente por el ordenador. De esta forma, solo los escribimos una vez y los podemos volver a utilizar cuantas veces querramos. Comencemos, pues!

Lo primero que necesitamos es un pensar que debemos ir haciendo alguna operación un archivo a la vez. Lo primero que viene a nuestra mente es algo cíclico! En ese caso, necesitamos utilizar un comando `for` que vaya accesando cada archivo a la vez.

```
for archivo in listado de arhivos?
do
calcular las medias de alguna manera
done
```

Inmediatamente nos damos cuenta de que necesitamos el listado de los archivos que tengan que ver con agua. Cómo hacámos esto? Es cierto! Tenemos un comando `find` que nos permite hallar eso. Intentamos colocarlo dentro del código y ...

```
for archivo in $(find ../TQCA/Data/Agua*)
do
calcular las medias de alguna manera
done
```

Sí, debemos colocar nuestro comando entre paréntesis y con el signo `$` para que se genere la lista. Así como cuando extraíamos el valor de una variable, estamos extrayendo la lista. Genial, ahora ya tenemos la lista. Pero ... nuestra jefa probablemente la querrá ordenada. Hmm ... arreglemos eso.

```
for archivo in $(find ../TQCA/Data/Agua* | sort)
do
calcular las medias de alguna manera
done
```

Perfecto! Ahora, antes de comenzar a calcular las medias de los parámetros de cada archivo, sería conveniente mostrar de qué archivo es que se está calculando la media. Agregemos entonces, antes de cada dato, el nombre del archivo.

```
for archivo in $(find ../TQCA/Data/Agua* | sort)
do
echo $archivo
calcular las medias de alguna manera
done
```

Ahora sí, a calcular las medias. Esto se supone que lo hace la herramienta *med.sh* Pero ... qué es realmente esa herramienta? Pues esto, para nuestra sorpresa, es **otro** script. Resulta que para calcular las medias de los parámetros en un archivo con resultados de un muestreo de aguas, el script se debe de ejecutar de la siguiente forma.
`bash med.sh nombre_del_archivo.csv`

Qué es lo que está sucediendo acá? `bash` es el comando que *activa* o *ejecuta* ese script. Luego viene el nombre del script a ejecutar, y termina con el nombre del archivo del cual se está realizando los cálculos. Al implementar esto dentro de nuestro script (sí, se puede correr un script desde otro script), el código se va a ver algo así:

```
for archivo in $(find ../TQCA/Data/Agua* | sort)
do
echo $archivo
echo $(bash med.sh $archivo)
done
```

Ya casi terminarmos. Ahora, sería conveniente saber cuáles son los datos que se está calculando en cada columna. Para eso, agregamos otra línea más extrayendo del contenido de cada archivo la primera fila, y de ella los nombres de las columnas excepto la primera que sabemos que son los nombres de los lugares.

```
for archivo in $(find ../TQCA/Data/Agua* | sort)
do
echo $archivo
echo $(cat $archivo | head -1 | cut -d , -f 2-)
echo $(bash med.sh $archivo)
done
```

Excelente, terminamos el script. Pero momento, si se trata de muchos archivos, no podemos esperar a que nos muestre todo en pantalla. Eso sería poco práctico, pues tendríamos que copiárselo y pegárselo en un editor de texto, o peor aún, en el correo electrónico. Para ello, mejor crear un archivo nuevo, e ir agregando los resultados a él. Para crear el archivo nuevo rápidamente, solo introduciremos algún texto con el comando `>` . Y posteriormente iremos agregando cada línea al archivo en vez de mostrarla en pantalla. Esto lo haremos con el comando `>>` Este comando no solo introduce texto en un archivo, como su compañero, sino que lo agrega al texto ya existente. Procedamos entonces.

```
echo Medias de los muestreos en el lago de Atitlan > resultados.txt
for archivo in $(find ../TQCA/Data/Agua* | sort)
do
echo $archivo >> resultados.txt
echo $(cat $archivo | head -1 | cut -d , -f 2-) >> resultados.txt
echo $(bash med.sh $archivo) >> resultados.txt
done
```

Finalizamos! Ahora que ya tenemos la idea completa, abrimos nano y escribimos el script, lo guardamos con el nombre que más nos guste (y extensión *sh*) y lo ejecutamos escribiendo `bash nombre_de_mi_script.sh` Al final nos dirigimos a nuestra carpeta de trabajo en donde hallamos el archivo *resultados.txt*, el cual le enviamos a la investigadora principal por correo. Todo salión en tiempo y ahora ya, sino vuelve a pedir esos datos dentro de los próximos 3 meses, solo tendremos que correr el script nuevamente; la cantidad de archivos/muestreos realizados no importa, pues todo ya está automatizado.

1.8. Comentarios Finales

Felicidades, has completado el primer día del taller de QCA. Ahora ya conoces el alcance y las posibilidades de lo que se puede hacer en una línea de comando de Linux. Esta es, según muchos, la parte que aparenta ser más complicada y oscura. Inténtalo, falla y vuelve a intentarlo sin miedo. Nadie es perfecto la primera vez.

Si deseas profundizar, lee, experimenta y sigue escribiendo scripts. Para comenzar quizá sea bueno que revises qué es lo que hace *med.sh*. Te recomiendo leer los manuales de cada comando e ir probando qué hace cada uno. Después de todo, si tus scripts son archivos independientes, no puedes romper nada ni desperdiciar mucho. Al contrario, hay mucho que puedes aprender.

Felicidades nuevamente, y nos vemos mañana en la introducción a bases de datos!

1.9. Glosario de comandos sencillos

- **man**: muestra páginas o documentos de ayuda sobre un comando en particular. Para salir de la página de ayuda, presionar *q*.
- **whoami**: muestra al usuario activo.
- **pwd**: muestra el directorio en el que se está trabajando actualmente.
- **ls**: mostrar (listar) el contenido de un directorio.
- **cd**: cambia de directorio al directorio que se ponga a continuación, o regresa al directorio *home* si no se agrega nada.
- **mkdir**: crea un nuevo directorio con el nombre que se ponga a continuación.
- **rm**: elimina el archivo que se coloque a continuación.

- **rmdir**: elimina el directorio que se coloque a continuación. El directorio debe de estar vacío.
- **nano**: editor de texto en línea de comando. El nombre que se coloque a continuación será el nombre del archivo.
- **cp**: copia un archivo de un lugar a otro. Se coloca a continuación el nombre, con ubicación, del archivo a copiar y después la ubicación en donde se colocará la copia.
- **mv**: corta y pega un archivo de una ubicación a otra. Para ello hay que colocar a continuación el nombre, con ubicación, del archivo a cortar, y el directorio, con el nombre del archivo, a donde se va a pegar.
- **wc**: cuenta la cantidad de líneas y de palabras en un archivo de texto.
- **chmod**: cambia los permisos de lectura, escritura o ejecución de un documento.
- **chown**: cambia el usuario y/o el grupo al que el documento pertenece.
- **ln**: crea vínculos duros. Para crear vínculos simbólicos se agrega la bandera *-s*.
- **sort**: ordena una lista de palabras en orden alfabético.
- **cat**: muestra todo el contenido de un archivo de texto.
- **head**: muestra las primeras líneas de un archivo de texto.
- **tail**: muestra las últimas líneas de un archivo de texto.
- **uniq**: elimina los duplicados adyacentes en una lista de palabras.
- **grep**: busca dentro de los archivos por una frase o palabra en particular. Esta se le tiene que dar posterior al comando.
- **find**: busca un archivo cuyo nombre tenga una frase o palabra en particular. Esta se le tiene que dar posterior al comando.
- **cut**: segmenta y filtra el texto tabulado de un archivo.
- **for**: crea ciclos que se mueven a lo largo de una lista dada de elementos.

Licencia