La línea de comandos

NOTA: Esta *lecture* está basada parcialmente en las notas del curso C4P de California Polytechnic State University, San Luis Obispo, y las notas del Dr. Adolfo de Unánue Tiscareño, con su permiso.

Introducción

El `shell` de Unix (en su caso particular es un `shell` de `GNU/Linux`), es más viejo que todos nosotros. Y el hecho de que siga activo, y en uso, se debe a que es una de las invenciones humanas más exitosas para usar la computadora de manera eficiente.

De una manera muy rápida el `shell` puede hacer lo siguiente:

- Un intérprete interactivo: lee comandos, encuentra los programas correspondientes, los ejecuta y despliega la salida.
  - Esto se conoce como **REPL**: *Read, Evaluate, Print, Loop*
- La salida puede ser redireccionada a otro lugar además e la pantalla. (Usando `>` y `<`).
- Una cosa muy poderosa (y en la que está basada --como casi todo lo actual--) es combinar comandos que son muy básicos (sólo hacen una sola cosa) con otros para hacer cosas más complicadas (esto es con un *pipe* `|`).
- Mantiene un histórico que permite rejecutar cosas del pasado.
- La información es guardada jerárquicamente en carpetas o directorios.
- Existen comandos para hacer búsquedas dentro de archivos (`grep`) o para buscar archivos (`find`) que combinados pueden ser muy poderosos.
  - Uno puede hacer *data analysis* solamente con estos comandos, así de poderosos son.
- Las ejecuciones pueden ser pausadas, ejecutadas en el *fondo* o en máquinas remotas.
- Además es posible definir variables para usarse por otros programas.
- El `shell` cuenta con todo un lenguaje de programación, lo que permite ejecutar cosas en **bucles**, **condicionales**, y hasta cosas en paralelo.

La computadora desde cerca

Al final las computadoras sólo hacen cuatro cosas:

- Ejecutan programas
- Almacenan datos
- Se comunican entre sí para hacer las cosas recién mencionadas.
- Interactúan con nosotros.
  - La interacción puede ser gráfica (como están acostumbrados) conocida también como **GUI** (*Graphical User Interface*) vía el ratón u otro periférico, o desde la línea de comandos, llamada como **CLI** (*Command Line Interface*).

La línea de comandos

La línea de comandos es lo que estará entre nosotros y la computadora casi todo el tiempo en este curso.

La **CLI** es otro programa más de la computadora y su función es ejecutar otros comandos. El más popular es `bash`, que es un acrónimo de *Bourne again shell*. Aunque en esta clase también usaremos `zsh`.

Archivos y directorios

La computadora guarda la información de una manera ordenada. El sistema encargado de esto es el `file system`. Básicamente es un árbol de información (aunque hay varios tipos de `file systems` que pueden utilizar modificaciones a esta estructura de datos, lo que voy a decir aplica desde su punto de vista como usuarios) que guarda los datos en una abstracción que llamamos *archivos* y ordena los archivos en *carpetas* o *directorios*, los cuales a su vez pueden contener otros *directorios*.

Muchos de los comandos del **CLI** o `shell` tienen que ver con la manipulación del `file system`.

Ejercicio:

- Inicia una sesión en `docker`
- Deberías de ver algo como esto:

```
root@6ddfd57cc4ec:/#
```

- Teclea `whoami` y luego presiona *enter*. Este comando te dice que usuario eres. Observa que el usuario actual es `root`. Este usuario es un **superusuario**, el tiene poderes para modificar todo, obvio, esto es peligroso, por lo que será mejor cambiar de usuario, en particular al usuario `jovyan`.

- Teclea `su jovyan`. ¿Qué pasó?

- Comprueba que eres el usuario `jovyan` ¿Cómo le podrías hacer?

- Para saber donde estamos en el `file system` usamos `pwd` (de *printing working directory*).

    - Estamos posicionados en la raíz del árbol del sistema, el cual es simbolizada como `/`.

- Para ver el listado de un directorio usamos `ls`

    - Ahora estás observando la estructura de directorios de `/`.

- Los comandos (como `ls`) pueden tener modificadores o *banderas*, las cuales modifican (vaya sorpresa) el comportamiento por omisión del comando. Intenta con `ls -l`, `ls -a`, `ls -la`, `ls -lh`, `ls -lha`. Discutan las diferencias entre cada bandera.

- Para obtener ayuda puedes utilizar `man` y el nombre del comando. ¿Cómo puedes aprender que hace `ls`?

- Otro comando muy útil (aunque no lo parecerá ahorita) es `echo`.

- Las variables de sistema (es decir globales en tu sesión) se pueden obtener con el comando `env`. En particular presta atención a `HOME`, `USER` y `PWD`.

- Para evaluar la variable podemos usar el signo de moneda `$`,

  - Imprime las variables con `echo`, e.g. `echo $USER`.

- El comando `cd` permite cambiar de directorios (¿Adivinas de donde viene el nombre del comando?) La sintáxis es `cd nombre_directorio`. ¿Puedes navegar hasta tu `$HOME`?

- ¿Qué hay de diferente si ahí ejecutas `ls -la`?

  - Las dos líneas de hasta arriba son `.` y `..` las cuales significan *este directorio* (`.`) y el directorio padre (`..`) respectivamente. Los puedes usar para navegar (i.e. moverte con `cd`)
  - ¿Puedes regresar a raíz?
  - En raíz ¿Qué pasa si ejecutas `cd $HOME`?
  - Otras maneras de llegar a tu `$HOME` son `cd ~` y `cd` solito.

- Verifica que estés en tu directorio (¿Qué comando usarias?) Si no estás ahí, ve a él.

- Para crear un directorio existe el comando `mkdir` que recibe como parámetro un nombre de archivo.

  - Crea la carpeta `test`. Entra a ella. ¿Qué hay dentro de ella?

- Vamos a crear un archivo de texto, para esto usaremos **GNU Emacs**. **GNU Emacs** es un editor de textos muy poderoso. Lo aprenderemos en la clase (y quizá en algún seminario). Por el momento teclea `emacs hola.txt` y presiona enter (la primera vez que lo corras puede tardar mucho, está instalando toda la configuración para la clase).

- Aparecerá una barra de menú abajo. Esto indica que ya estás en **GNU Emacs**. Teclea "¡Hola Mundo!" y luego presiona la siguiente
  combinación de teclas: `Ctrl+x` seguido de `Ctrl+s` (guardar cambios). Ahora presiona `Ctrl-x` y luego `Ctrl-c` (salir de emacs). Esto los devolverá a la **CL**.

- Verifica que esté el archivo.

- Para borrar usamos el comando `rm` (de *remove*),¿Cómo crees que se borraría un directorio?

- Borra el archivo `hola.txt`.

- ¿Ahora puedes borrar el directorio `test`? ¿Qué falla? ¿De dónde puedes obtener ayuda?

- Crea otra carpeta llamada `tmp`, crea un archivo `copiame.txt` con emacs, escribe en él: "Por favor cópiame".

- Averigua que hacen los comandos `cp` y `mv`.

- Copia el archivo a uno nuevo que se llame `copiado.txt`.

- Borra `copiame.txt`.

- Modifica `copiado.txt`, en la última línea pon "¡Listo!".
- Renombra `copiado.txt` a `copiame.txt`.
- Por último borra toda la carpeta `tmp`.

Navegar

Moverse rápidamente en la **CLI** es de vital importancia. Teclea en tu **CLI**

```
Anita lava la tina
```

Y ahora intenta lo siguiente:

- `Ctrl + a` Inicio de la línea
- `Ctrl + e` Fin de la línea
- `Ctrl + r` Buscar hacia atrás
  - Elimina el *flechita arriba*
- `Ctrl + b` / `Alt + b`
- `Ctrl + f` / `Alt + f`
- `Ctrl + k` - Elimina el resto de la línea (en realidad corta y pone en el búfer circular)
- `Ctrl + y` - Pega la último del búfer.
- `Alt + y` - Recorre el búfer circular.
- `Ctrl + d` - Cierra la terminal
- `Ctrl + z` - Manda a *background*
- `Ctrl + c` - Intenta cancelar

Pipes y flujos

- `|` (pipe) “Entuba” la salida de un comando al siguiente
- `>`, `>>`, Redirecciona la salida de los comandos a un sumidero.

```
ls >> prueba.dat
```

- `<` Redirecciona desde el archivo

```
sort < prueba.dat # A la línea de comandos acomoda con sort,
sort < prueba.dat > prueba_sort.dat # Guardar el sort a un archivo.
```

- `&&` es un AND, sólo ejecuta el comando que sigue a `&&` si el primero es exitoso.

```
> ls && echo "Hola"
> lss && echo "Hola"
```

Otros comandos

- `wc` significa *word count*

    - Cuenta palabras,renglones, bytes, etc.

- En nuestro caso nos interesa la bandera `-l` la cual sirve para contar líneas.

```
> wc -l /etc/passwd
24 /etc/passwd
```

- `head` y `tail` sirven para explorar visualmente las primeras diez (default) o las últimas diez (default) renglones del archivo, respectivamente.

```
> head /etc/passwd
> tail -3 /etc/passwd
```

- `cat` concatena archivos y/o imprime al `stdout`

```
> echo 'Hola mundo' >> test
> echo 'Adios mundo cruel' >> test
> cat test
...
> cp test test2
> cat test test2 > test3
> wc -l test*
```

Existen otros comando poderosos como `split` , `uniq` , `grep` , etc.

Con estos ejercicios deberías de ser capaz de manejar los básicos del `file system` y de la línea de comandos

Antes de partir… ¿Por qué usamos `zsh` en lugar de `bash` ?

`zsh` es un `bash` recargado, para saber que puede hacer revisa esto y esto.

Además, en tu `docker` el `zsh` viene recargado con `oh-my-zsh` .