Lectures

Command line

La computadora

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*).

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.

¿Por qué?

En muchas ocasiones, se verán en la necesidad de responder muy rápido y en una etapa muy temprana del proceso de **big data** . Las peticiones regularmente serán cosas muy sencillas, como estadística univariable y es aquí donde es posible responder con las herramientas mágicas de UNIX.

Línea de comandos

La línea de comandos es lo que estará entre nosotros y la computadora casi todo el tiempo en este curso. De hecho, una lectura obligada (no me hagan que la deje de tarea es In the beginning…was de command line de **Neal Stephenson**, el escritor de **Criptonomicon**.
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`.

Command Line 101

La primera regla de la línea de comandos es: *ten cuidado con lo que deseas, no se te vaya a cumplir*. La computadora hará exactamente lo que le digas que haga, pero recuerda que humanos (probablemente) tienen dificultadas para expresarse en *lenguaje de computadoras*. Esta dificultad puede ser muy peligrosa, sobre todo si ejecutas programas como `rm` (*borrar*) o `mv` (*mover*).
Puedes crear archivos *dummy* para este curso usando el comando `touch`:

```
touch space\ bars\ .txt
```

Nota que usamos el caracter `\` para indicar que queremos un espacio en el nombre de nuestro archivo. Si no lo incluyes

```
touch space bars .txt
```

… la computadora creará tres archivos separados: `space`, `bars`, and `.txt`.

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`.

Conocer los alrededores

Navegación en la terminal

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

```
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` Mueve el cursor hacia atrás

`Ctrl + f` / `Alt + f` Mueve el cursor hacia adelante

`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* el programa que se está ejecutando

`Ctrl + c` Intenta cancelar

¿Dónde estoy?

`pwd` Imprime el nombre del *directorio* actual

`cd ..` Cambia el directorio un nivel arriba (a el directorio "padre"[1])

`cd ~` **/** `cd` Cambia el directorio `$HOME` (tu directorio)

¿Qué hay en mi directorio (*folder*) ?

`ls` Lista los contenidos (archivos y directorios) en el directorio actual, pero no los archivos *ocultos*.

`ls -l` Lista los contenidos en formato *largo* (`-l`), muestra el tamańo de los archivos, fecha de último cambio y permisos

`tree` Lista los contendios en el directorio actual y todos los sub-directorios en una estructura de *árbol*

`tree -L 2` Límite la expansión del *árbol* a dos niveles

`tree -hs` Muestra los archivosshows file sizes (`-s`) in human-readable format (`-h`)

¿Qué hay en mi archivo?

`head -n10 $f` Muestra el principio (*head*) del archivo, `-n` especifica el número de líneas (`10`).

`tail -n10 $f` Muestra la final (*tail*) del archivo.

`tail -n10 $f | watch -n1` Muestra la parte final del archivo cada segundo (usando `watch`)

`tail -f -n10 $f` "sigue"(*follows*) (`-f`) la parte final del archivo, cada vez que hay cambios

- Útil cuando estás ejecutando un programa que guarda información a un archivo, por ejemplo)

`wc $f` Cuenta las palabras, caracteres y líneas de un archivo

¿Dónde está mi archivo?

`find -name «lost_file_name>type f` Encuentra el archivo por nombre

`find -name «lost_dir_name>type d` Encuentra directorios por nombre

Caveats con `git`

Mover archivos puede confundir a `git`. Si estás trabajando con archivos en `git` usa lo siguiente:

- `git mv /source/path/$move_me /destination/path/$move_me`
- `git rm $remove_me`

[1] Lo siento, terminología … ¿Alguna sugerencia?

Ejercicio:

- Teclea `whoami` y luego presiona **enter**. Este comando te dice que

usuario eres.

- Teclea `cd /`
- Para saber donde estamos en el `file system` usamos `pwd` (de *print 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** (*flags*), las cuales modifican (vaya sorpresa) el comportamiento por omisión del comando. Intenta lo siguiente: `ls -l`, `ls -a`, `ls -la`, `ls -lh`, `ls -lha`. Discute con tu compañero junto a tí las diferencias entre las banderas.
- Para obtener ayuda puedes utilizar `man` (de *manual*) y el nombre del comando. ¿Cómo puedes aprender que hace `ls`?
- Puedes buscar dentro de `man page` para `ls` (o de cualquier otro manual) si tecleas `/` y escribiendo la palabra que buscas, luego presiona `enter` para iniciar la búsqueda. Esto te mostrará la primera palabra que satisfaga el criterio de búsqueda. `n` te mostrará la siguiente palabra. `q` te saca del `man page`.
- Busca la bandera para ordenar (*sort*) el listado de directorios por tamaño.
- Muestra el listado de archivos de manera ordenada por archivo.
- 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` (sin argumento).
- 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 `nano`[2]. Por el momento

teclea

```
nano hola.txt
```

y presiona enter.

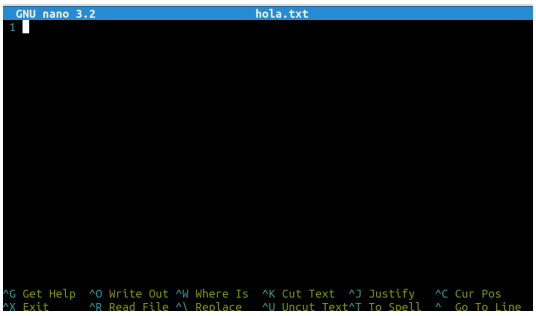

Figura 1: Editor `nano`, mostrando el archivo recién creado `hola.txt` .

- Teclea

```
¡Hola Mundo!
```

y luego presiona la siguiente combinación de teclas: `Ctrl+O` para guardar el archivo (Te va a preguntar sobre *dónde* guardarlo).

- `Ctrl+X` para salir de `nano`. Esto los devolverá a la línea de comandos.
- Verifica que esté el archivo.
- Para ver el contenido de un archivo tienes varias opciones (además de abrir `nano`): `cat`, `more`, `less`

cat `hola.txt`

- Para borrar usamos el comando `rm` (de *remove*).
- Borra el archivo `hola.txt`.
- ¿Cómo crees que se borraría un directorio?
- ¿Ahora puedes borrar el directorio `test`? ¿Qué falla? ¿De dónde puedes obtener ayuda?
- Crea otra carpeta llamada `tmp`, crea un archivo `copiame.txt` con `nano`, 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`.

[2]Otras opciones son: **GNU Emacs** (un editor de textos muy poderoso. Es el que estoy usando) o **vi**. No importa cual escojas, aprende a usarlo muy bien. Recuerda, queremos disminuir el dolor.

Wildcars

La línea de comandos permite usar comodines (*wildcards*) o expresiones regulares (*regular expressions*) para encontrar archivos:
Listar todos los archivos que tienen una extensión `txt`

```
ls *.txt
```

Listar todos los archivos que contienen `a` en el nombre y extensión `txt`

```
ls *a*.txt
```

Listar todos los archivos que tienen 5 caracteres en el nombre:

```
ls ?????,txt
```

Datos para jugar

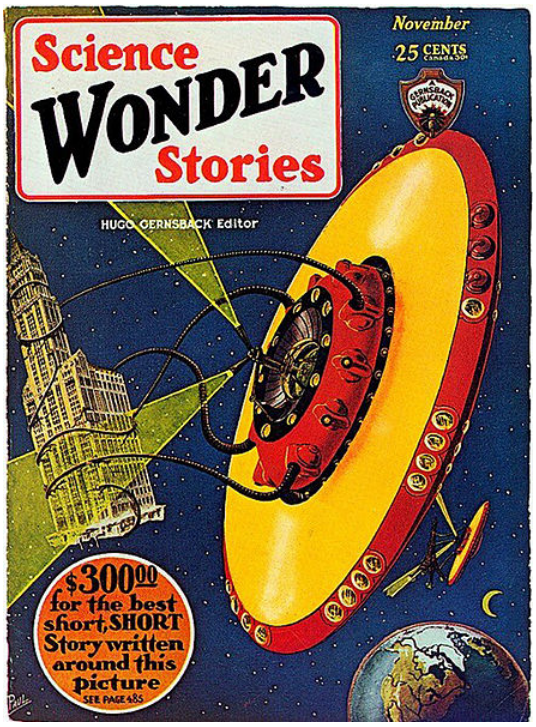

Figura 2: Portada de la revista *Science Wonder Stories* (1929) Imagen tomada de **Wikipedia**

- Para los siguientes ejemplos trabajaremos con los archivos encontrados en **The National UFO Reporting Center Online Database**.
- Estos datos representan los *avistamientos* de OVNIS en EUA.
- Usaremos como ejemplo la descarga el mes de Diciembre y Noviembre de 2014 http://www.nuforc.org
- Se encuentra en la carpeta `data`.

Conectando comandos

Entubando

`|` **(*pipe*)** "Entuba" la salida de un comando al siguiente comando.

```
ls -la | wc -l

# grep "busca y selecciona" cadenas o patrones (lo veremos al rato)
seq 50 | grep 3
```

Redireccionando *hacia*

Redirecciona la salida de los comandos a un sumidero (e.g. un archivo, o la pantalla o la impresora).

```
ls >> prueba.dat

seq 10 > numeros.txt
```

Redireccionando *desde*

< 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.
```

Ejecución condicional

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

```
ls && echo "Hola"

lss && echo "Hola"
```

Ejercicio

Transforma el archivo de `data` de tabuladores a `|`, cambia el nombre con terminación `.psv`.

Algunos comandos útiles

seq

Genera secuencias de números

```
seq 5
seq inicio step final
seq 1 2 10
```

Usando otro separador (`-s`) que no sea es caracter de espacio

```
seq -s '|' 10
```

Agregando *padding*

```
seq -w 1 10
```

tr

Cambia, reemplaza o borra caracteres del `stdin` al `stdout`

```
echo "Hola␣mi␣nombre␣es␣Adolfo␣De␣Unánue" | tr '[:upper]' '[:lower]'

echo "Hola␣mi␣nombre␣es␣Adolfo␣De␣Unánue" | tr -d ' '

echo "Hola␣mi␣nombre␣es␣Adolfo␣De␣Unánue" | tr -s ' ' '_'
```

wc

- `wc` significa *word count*
  - Cuenta palabras,renglones, bytes, etc.
- En nuestro caso nos interesa la bandera `-l` la cual sirve para contar líneas.

```
seq 30 | grep 3 | wc -l
```

Ejercicio

¿Cuantos avistamientos existen Noviembre, Diciembre 2014?

```
wc -l UFO-Dic-2014.tsv
```

```
wc -l *
```

head, tail

- `head` y `tail` sirven para explorar visualmente las primeras diez

(*default*) o las últimas diez (*default*) renglones del archivo, respectivamente.
```
>cd data
>head UFO-Dic-2014.tsv
>tail -3 UFO-Dic-2014.tsv
```

cat

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

```
echo 'Hola mundo' >> test

echo 'Adios mundo cruel' >> test

cat test

rm test
```

También podemos concatenar archivos

```
cat UFO-Nov-2014.tsv UFO-Dic-2014.tsv > UFO-Nov-Dic-2014.tsv
wc -l UFO-Nov-Dic-2014.tsv
```

En el siguiente ejemplo redireccionamos al `stdin` el archivo como entrada del `wc -l` sin generar un nuevo proceso

```
< numeros.txt wc -l
```

split

- `split` hace la función contraria de `cat`, divide archivos.
- Puede hacerlo por tamaño (bytes, `-b`) o por líneas (`-l`).

```
split -l 500 UFO-Nov-Dic-2014.tsv
wc -l UFO-Nov-Dic-2014.tsv
```

cut

- Con `cut` podemos dividir el archivo pero por columnas.
- Donde columnas puede estar definido como campo (`-f`, `-d`), carácter (`-c`) o bytes (`-b`).
- En este curso nos interesa partir por campo.

Creemos unos datos de prueba

```
echo "Adolfo|1978|Físico" >> prueba.psv
echo "Patty|1984|Abogada" >> prueba.psv
```

Ejecuta los siguientes ejemplos, ¿Cuál es la diferencia?

```
cut -'d'| -f1 prueba.psv
cut -'d'| -f1,3 prueba.psv
cut -'d'| -f1-3 prueba.psv
```

Ejercicio

¿Qué pasa con los datos de avistamiento? Quisiera las columnas 2, 4, 6 ó si quiero las columnas Fecha, Posted, Duración y Tipo (en ese orden).
¿Notaste el problema? Para solucionarlo requeriremos comandos más poderosos…
Lee la documentación (`man cut`), ¿Puedes ver la razón del problema?

uniq

- `uniq` Identifica aquellos renglones consecutivos que son iguales.
- `uniq` puede contar (`-c`), eliminar (`-u`), imprimir sólo las duplicadas (`-d`), etc.

sort

- `sort` Ordena el archivo, es muy poderoso, puede ordenar por

columnas (`-k`), usar ordenamiento numérico (`-g`, `-h`, `-n`), mes (`-M`), random (`-r`) etc.

```
sort -t "," -k 2  UFO-Nov-Dic-2014.tsv
```

uniq y sort

- Combinados podemos tener un `group by`:

```
# Group by por estado y fecha
cat UFO-Dic-2014.tsv |\
    cut -d$'\t' -f1,3 |\
    tr '\t' ' ' |\
    cut-d' ' -f1,3 |\
    sort -k 2 -k 1 |\
    uniq -c |\
    sort -n -r -k 1 |\
    head
```

Ejercicio

- ¿Cuál es el top 5 de estados por avistamientos?
- ¿Cuál es el top 3 de meses por avistamientos?

Otros comandos útiles

- `file -i` Provee información sobre el archivo en cuestion

```
file -i UFO-Dic-2014.tsv
```

- `iconv` Convierte entre encodings, charsets etc.

```
iconv -f iso-8859-1 -t utf-8 UFO-Dic-2014.tsv  > UFO-Dic_utf8.tsv
```

- `od` Muestra el archivo en octal y otros formatos, en particular la bandera `-bc`

lo muestra en octal seguido con su representación ascii. Esto sirve para identificar separadores raros.

```
od -bc UFO-Nov-2014.tsv    | head -4
```