**2019.11.15 INTRO + GIT + COMMAND LINE**

Proyecto de master serán unas 50-80 horas.

**The data science process:**

* Ask and interesting question.
* Get the data.
* Explore the data.
* Model the data: darle forma para que coincida con la manera en que quieres analizar los datos.
* Communicate and visualize the results.

**Qué tipo de preguntas podemos resolver?**

MACHINE LEARNING & STATISTICS APPLICATIONS:

* Supervised Learning: classification, regression, ranking
* Unsupervised learning: clustering, association mining, segmentation, dimension reduction
* Reinforcement learning: decision process, rewards system, recommendation systems. (**No lo vemos en el master).**

**NOTA**: Subir código a Github todas las semanas y compartir para crear portfolio y tenerlo ahí cuando vayas a buscar trabajo.

Hay que poner el enlace de github en el CV.

Es importante que se vea un commit semanal, no hacerlo todo al final. Subir lo que se dé cada semana en clase a Git para que el recruiter vea todo el trabajo hecho en meses.

Ver blog/post de Amadeus data scientist

<https://amadeus.com/en/insights/blog/ten-things-to-look-for-when-hiring-a-data-scientist>

<http://www.amadeus.com/blog/tag/data-scientist/>

<https://www.avanzaentucarrera.com/orientacion/profesiones/digitales/>

cd ..

mkdir : make directory

mkdir git: crear directorio llamado dir

cd : change directory ir al directorio home

cd /home/angelrps: me lleva a la ruta marcada.

cd .. subir un nivel de carpetas

cd ../.. : subir dos niveles de carpetas

echo: mostrar en pantalla un texto o una variable

cat README.md : muestra lo que contiene el archivo README.md dentro

TRABAJANDO CON GIT EN LINEA DE COMANDOS

Primero crear repositorio nuevo desde Github

Crear ese directorio en nuestra carpeta local (mkdir nombrerepositorio)

**git init** : convierte la carpeta actual en un repositorio git

**git remote add origin https://github.com/angelrps/master\_kschool\_tst.git**

le dice a git añademe este repositorio local a este repositorio remote en githiub. Hay que hacerlo solo una vez al empezar a trabajar con el repositorio. Si no se hace no se puede hacer push.

**git add “nombre de archivo”** : añade un archivo al repositorio

**git commit –m** “Texto explicativo sobre el commit” : git hace una foto del contenido del repositorio y añade un comentario explicativo de qué ha cambiado en ese commit. Pero ojo, esto pasa en el servidor local.

**Git push origin master** : para enviar el trabajo hecho a la rama master (que siempre existe por defecto). El push es el que manda el trabajo local al remoto. “origin” es la conexión en este caso github. Master es la rama. Se debe haber configurado el remote origin previamente.

Esto de abajo hay que hacer una vez para que git sepa quien eres:

Git config –global user.name “Angel Ruiz-Peinado”

Git config –global user.email [angelruizpeinado@gmai.com](mailto:angelruizpeinado@gmai.com)

**rm –rf directory** : eliminar directorio y todo lo que hay dentro

**git remote show origin** : muestra el repositorio remote al que estás haciendo push y pull.

**git ls-files** : show all files being tracked

**git checkout** <rama o master> : establece esa rama o master como actual

**rm –rf ./.git** : I don´t what this to be a local git repository

**git reset –hard HEAD** = si has hecho algo sin querer con esto reseteas hasta el último commit

**2019.11.16 COMMAND LINE**

**Mirar si instalar Shell zsh en lugar de bash**

Data Science challenge: prueba que hace Amadeus para entrar. Igor nos avisará con 10 días de antelación por si lo queremos hacer.

Cuándo se usa la línea de comandos:

Raw Data: conectarse a datos de origen confidenciales como las reservas de una aerolínea y sacar parte de ellos para analizar.

Structure Date:

Cuando hablamos de Linux hablamos de UNIX:

Command line tools

Terminal: pantalla negra donde se escribe

Shell: lo que interpreta el comando

Operating System: se llama kernel

Qué es GNU vs Linux:

Linux es el kernel

GNU es todo: command line + terminal + Shell + kernel (que cogieron el de linux)

GIT:

Antes de empezar a trabajar hacer primero un pull para tener la última versión actualizada.

Basic commands:

* Clone

**LINEA DE COMANDOS**

Primero se escribe comando y luego sus parámetros.

**echo**: devuelve lo que viene después

**pwd**: print working dir, muestra la ruta en la que te encuentras

**$:** significa que después viene una variable

**echo $SHELL**: devuelve la Shell en la que estoy. Lo haces cuando te conectas a una máquina desconocida para ver en qué Shell estás trabajando.

**echo $PWD**: devuelve el valor de la variable

**cat** : devuelve el contenido entero de un fichero.

**cat –n**: añade el número de línea al lado.

**whoami**: este comando te devuelve tu nombre de usuario.

**cat /etc/os-release**: para saber qué distribución de Linux estamos usando. este fichero está en todos los shells.

**cat /proc/cpuinfo**: nos da información del cpu.

**/** : significa que vamos a dar un path absoluto

**./** : current directory

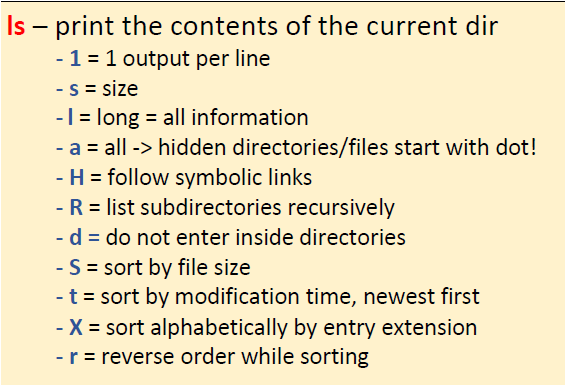
**../** = upper directory

**-** = toogle between last two directories.

**.** = hidden directories/files start with dot

**ls** = list, listar los archivos que hay en la carpeta en la que te encuentras

Hay diferentes comandos con ls. Buscar en el pdf y hacer lista



**¿cómo crear una variable?**

MASTER = Kschool

**echo $MASTER** : me muestra “kschool”

**“ ”** Comillas dobles agrupan varios parámetros en uno e interpretan los parámetros contenidos

**‘ ’** Comillas simples **no** interpretan el valor de los parámetros contenidos, interpretan lo de dentro como texto.

Acento inverso ´$MASTER´: ejecutan el comando que está debajo

Para ejecutar varios comandos en la misma línea usamos punto y coma ;

Echo; date : los dos son comandos

$ + () = ejecutar la variable que hay dentro del paréntesis

Echo “$(date)” : ejecuta el comando date. Ojo una variable (llamada por $) puede tener un comando dentro o no

Osea que hay varias maneras de ejecutar una variable. Intentar resumirlas juntas.

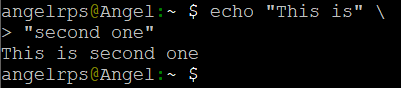
Como introducir una nueva línea: (varias formas)

Echo “lkshdjflkkhsd <enter>

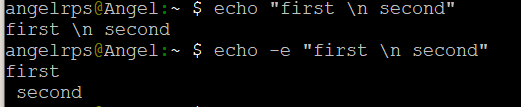
Continuoescribiendo” (cierro comillas y enter)

Echo “This is” \ <barra inversa y enter. La barra inversa no manda al Shell la info después de enter>

\n es nueva linea



Si no funciona poner -e



\t es tabulador

Para ignora una barra inversa hay que poner otra barra delante



Cd - : te lleva al directorio anterior en el que has estado

Cd ./ : mirar desde donde estamos. Para asegurarte que seleccionas el folder dentro deñl folder en que estás.

Cd . : muestra los directorios ocultos del folder donde estás

Osea para moverme a un directorio oculto pongo: cd .config <siendo config oculto>

Cat : muestra contenido del archive

Cat –n : lo muestra enumerando las líneas. Si pongo varios ficheros unos detrás de otro concatena su contenido.

Cat \* + tab muestra contenido de todos los ficheros

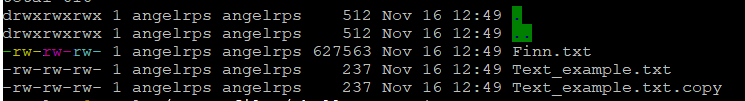
Cat \*.txt : muestra todos los ficheros .txt

Cat \*.txt\* : que incluya txt

Cat \*dk\* : todos los ficheros cuyo nombre incluye dk

**PERMISOS: r**ead, **w**rite, **e**xecute.

Chmod : comando para cambiar permisos (user, group, others)



Chmod u-w README.md : quito el permiso de write a user del archivo README

Chmod u+w,o-x README.md : doy permiso write a users y quitar execute a others.

Chmod 777 : dar permisos a todos en binario

**CREAR FILES / DIRECTORIES**

Crear directorio + subdirectorio a la vez. –p es la clave

**mkdir** –p two/three/four

Si los parent directories existen no hace falta la -p

**touch** newfile = Para crear un fichero llamado newfile

**COPY FILES/DIRECTORIES**

Para copiar es cp y ponemos origen y destino

SINTAX: ***cp [options] source destination***

**cp** file file.copy

Copiar archivo a otro directorio

Copiar varios ficheros a un directorio. El ultimo parámetro será el directorio, los del medio los ficheros

**–r** copiamos todos los directorios de dentro de un directorio a otro

**-p** (preserve) para preservar todos los permisos de un archivo cuando lo copiamos

**MOVE FILES/DIRECTORIES**

**mv <directory> ../** Mover un directorio un nivel por encima

**RENAME FILES/DIRECTORIES**

**mv four four\_new:** Mover un directorio y cambiarle el nombre. En realidad, no lo hemos movido, es como cambiar el nombre.

Ahora sí mover y cambiar el nombre

**mv four and/new\_file** (add/ sería el destino)

**-f**, **--force** : ignore non-existent files and arguments, never prompt

**DELETE FILES/DIRECTORIES**

Para borrar rm + las rutas del fichero

**rm –r** : recursive model, borrar todo lo de dentro.

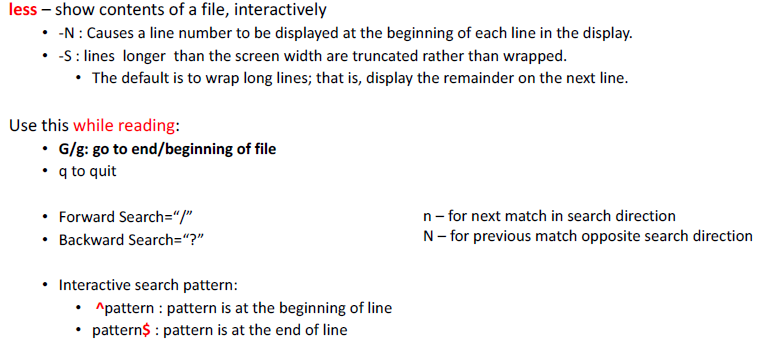
**rm –rf** (f es forzar, que no me pregunte si quiero eliminar ciertos ficheros)

**HELP UTILITIES**

**man** [comando] : man is short for manual. No todos los comandos tienen manual, en ese caso usar **–h** o **--help**.

**CONTENT UTILITIES**

**less** : muestra el texto de un archivo y permite interactuar con él.



**less** permite hacer búsqueda dentro del texto con barra / más el texto a buscar

Poniendo –N te pone el número de línea

**q** es para salir del less

Cuando se hace una búsqueda en less siempre se hace desde la línea primera de la pantalla no desde el principio del ficho

g vas al princpio del fichero y G vas al final

**COMMAND LINE TOOL TYPES**

type to find out the type of a command-line tool.

**HOW TO DEFINE ALIAS**

Alias son como shorcuts. Cómo crear un alias

alias p=’pwd’

alias p=”pwd”

Como añadir mis alias de un Shell a otro? No lo he pillado

**HOW TO KNOW WHICH BINARY ARE YOU EXECTING?**

**which**

**HOW TO KNOW THE LOCATION OF THE BINARY?**

**whereis**

**EXIT TERMINAL**

**ctrl + d** salgo del Terminal pero la línea actual debe estar vacía

**CONTROLLING COMMAND LINE**

**alt + b** = move backward word by word

**alt + f** = move forward word by word

**crtl + u** = cortar línea

**crtl + k** = cortar desde cursor hasta el final

**crtl + y** = pegar

**crtl + shift + c** = copy

**crtl + shift + v** = paste

**crtl + l** = clean screen

**crtl + r** = para buscar un comando ejecutado en el historial. Sigues pulsando crtl +r va pasando a través de todos los comandos que contienen tu búsqueda

**history** muestra todos los comandos ejecutados en la sesión.

**echo $HISTFILE**

**alt + c** = capitalize first letter of current of following Word.

**alt + u** = change the rest of the current Word of the following Word to uppercase.

**alt + l** = change the rest of the current Word of the following Word to lowercase.

**CONTENT UTILITIES**

**wc** (wordcamp) te da el número total de líneas, palabras y bytes



**wc -l**, --lines : líneas

**wc -w**, --words : palabras

**wc –c**, --bytes : bytes

**wc –m**, --chars : character counts.

cómo ver principio, final y contar palabras de un fichero.

**head** <fichero> = por defecto te saca primeras 10 líneas.

**head -5** <fichero> = te saca 5 líneas.

cuando después de head se pone –n el programa sabe que después va un numero

**head –n +5** muestras también las primeras 5 líneas

**head –n -5** muestra TODO excepto las últimas 5 líneas

**OJO** la diferencia entre head -5 y head –n -5!!!!

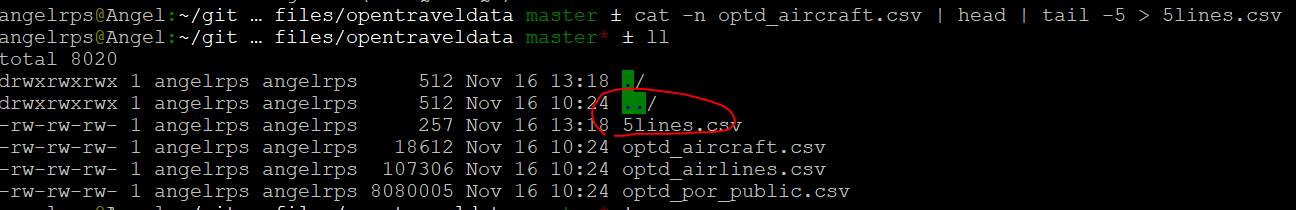
**tail** muestra el final de un documento. Y lo explicado antes sobre –n +-5 funciona al revés

**STANDARD INPUT / OUTPUT AND REDIRECTING**

**>** este símbolo graba texto en un fichero. Si no existe el fichero, lo crea.

**>>** doble símbolo lo añade al texto existente, sin borrar lo que hay.

**<** takes stdin from file



**THE PIPE**

Encadenar comandos con Pipe |

**FIND**

Comando find para buscar (con filtros)

SINTAC: **find [path] [conditions]**

Te devuelve todos los archivos por debajo del directorio donde estás

**find | wc** : número de líneas de la búsqueda para ver si es muy grande y se va a tirar dos años mostrando líneas

Primer parámetro siempre es el directorio donde comenzar la búsqueda

**find -name** \*text\_file\* = find by file name containing “text\_file”

**-iname** : ignore case

**-type d** coge solo directorio

**-type f** coge solo ficheros

**-maxdepth** 4 = máximo nivel de subfolders para buscar (1 is current directoy, 0 i s the command line)

**-mindepth** = minimum directoy levels.

**-perm** 777 = permisos, mirar ficheros con todos los permisos. Siempre seguido de un integer-

**¡-perm** = la exclamación invierte el resultado.

-empty tamaño 0

-size -10M menor de 10M

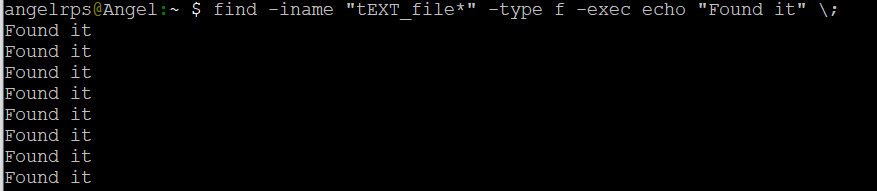
-size +10M mayor de 10M

Siempre poner – o + porque si no estoy buscando el tamaño EXACTO de 10M

-mmin -60 ficheros tocados en los últimos 60 minutos o menos

-mmin +60 ficheros tocados hace 60 minutos o más

-exec para ejecutar el input que tenga detrás



Ha executado la escritura de Found it tantas veces con archivos aha encontrado

{} para que ejecute el comando sobre cada fichero encontrado

**2019.11.22 COMMAND LINE ADVANCE**

**seq** crea una secuencia numérica

**seq [start][step][end]**

**seq 10** = start 1, step 1, end 10

seq 10 20 = start 10, step 1, end 20

**uniq** devuelve líneas únicas. Compara una línea con la siguiente, osea que para obtener un resultado real hay que ordenar los números.

**sort** ordena alfabéticamente

**sort –n** ordena numéricamente

**sort –n –u** ordena numéricamente y devuelve elementos únicos.

**sort -n uniq\_example.txt | uniq** = esto hace lo mismo

**uniq –d** (duplicates)= uniq tb te da los elementos duplicados

**uniq –c** (count) = número de elementos que se repite una línea

sort –nru (r de reverse) (u unique)= oirdenar de forma numérica inversa y devolder valores unicos.

***sort -t "^" -k 6r*** *optd\_aircraft.csv*

–t “^” = -t indica cual es el delimitador, en nuestro caso ^

-k = column

6 = columna 6

r = reverse. Ordenar en orden reverso.

***sort -t "^" -k 6,6*** *optd\_aircraft.csv*

**6,6** = desde columna 6 a la columna 6. Es decir, me quedo con una solo columna

Podemos ordenar sobre una columna y luego sobre la otra

***sort -t "^" -k6,6r -k8,8r*** *optd\_aircraft.csv | head*

el head de al final es para que me muestre solo 10 líneas

Cuál es el avión con más motores?

*sort -t "^" -k7,7r optd\_aircraft.csv | head*

Cuántos fabricantes diferentes tengo? (añado –u (unique) para darme los elementos únicos)

*sort -t "^" -k2,2r -u optd\_aircraft.csv | wc*

2 es la columna del fabricante. –u me devuelve las líneas únicas. wc me cuenta las líneas.

**CORTAR LINEAS**

**cut** –d “^” –f

-d = es el delimitador

-f = field, el equivalente a la –k del comando anterior.

***cut -d "d" -f 1,3-5 optd\_aircraft.csv | head***

saca las columas 1, 2, 3 y 5 del archive.

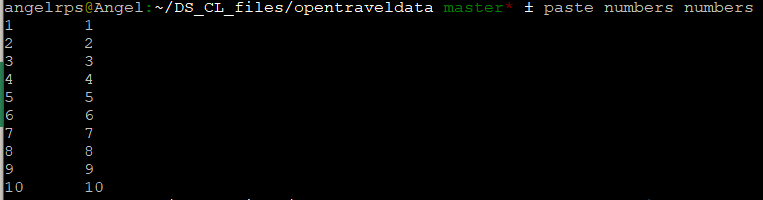
***cut -d "^" -f 1,3-5 --output-delimiter ";" optd\_aircraft.csv | head***

**--output-delimiter “;”** = cámbiame el delimitador del output de ^ a ;

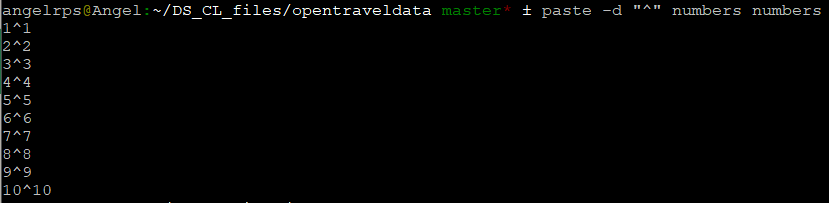
PASTE

paste [fichero] [fichero]

concatena líneas de dos ficheros en paralelo



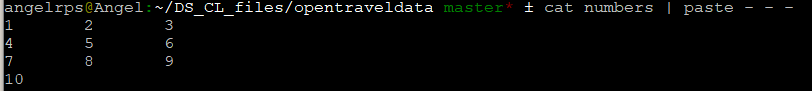
añadiendo delimitador



-s = serializa, convierte una columna en una línea.

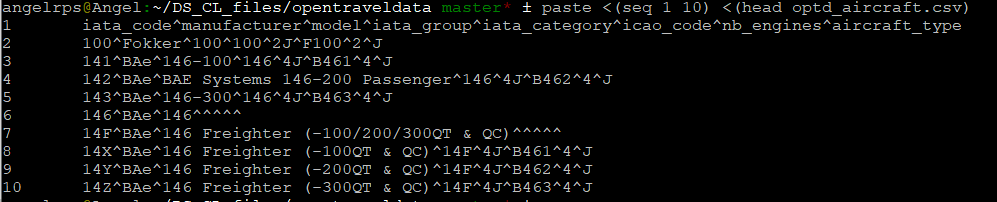


ordenar contenido en 3 columnas



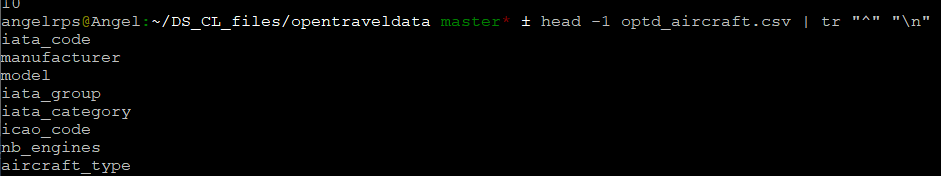
todo esto funciona bien cuando en la primera columna hay números, si hay letras el buffer da problemas.

para ejecutar comandos dentro del uso de paste usamos <() y el comando dentro



**tr** = comando **TRANSLATE**

cambiar delimitador de la primera línea a “\n”

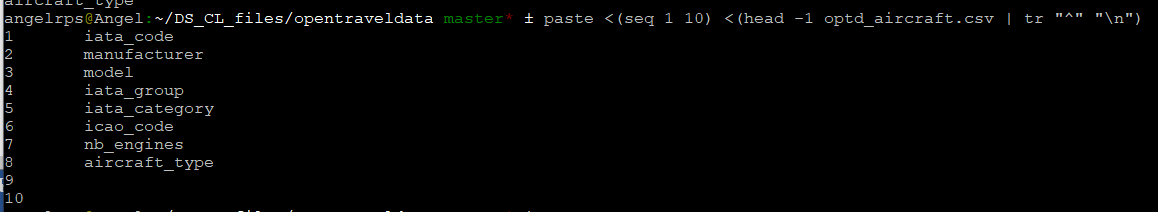


ejemplo más sencillo, cambia a minúscula con A mayúscula



usando distintos sets de entrada salida, busca carácter por carácter, no la palabra entera





-s squize = borra carácteres juntos duplicados

echo "msssssssaster data science" | tr -s "s"

**USAR SIEMPRE TRANSLATE PARA CAMBIAR DELIMITADORES**

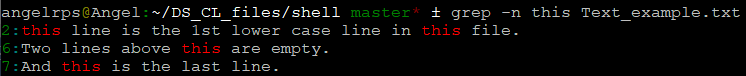
**cat optd\_aircraft.csv | tr "^" "\t" | head**

hay más opciones que no estoy pillando pero deben estar en la presentación

**grep**

devuelve la línea entera que contiene el string que buscamos. Es case sensitive. no dice cuántas veces se ha encontrado el string.

*grep -n this Text\_example.txt*



podemos poner un fichero detras de otro y te dice el fichero donde lo ha encontrado

**-v** = devuelve las líneas donde NO lo ha encontrado.

*grep -nv this Text\_example.txt*

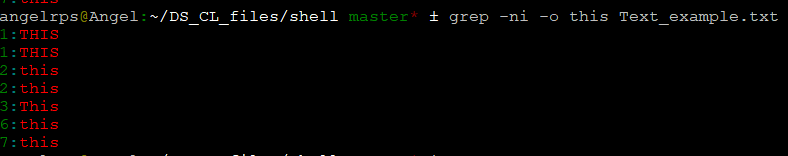
-i = ignora case sensitive

*grep -ni this Text\_example.txt*

**-c** = cuenta líneas donde se ha encontrado el string (OJO: no el número de string encontrados)

**-o**

= muestra el objeto encontrado



**sed STREAM EDITOR**

cambiar un string por otro (translate cambiaba un carácter por otro)

tiene muchas opciones.

**sed s/day/night**

s = substitute command

/ = delimitador , puede ser cualquier character.



la primera vez que encuentra un cambio lo cambia y se pasa a la otra línea. Para que siga buscando por toda la línea añadimos **global (g)**

*echo Sunday.day | sed s/day/night/g*

*Sunnight.night*

Hagamos el sed sobre un fichero

*sed s/this/WHHAT/g Text\_example.txt*

con I = ignore case (es una i mayúscula)

*sed s/this/WHHAT/gI Text\_example.txt*

-i = in place. Hace los cambios y los guarda en el fichero. Así que mejor hacer una copia!

*sed –i s/this/WHHAT/gI Text\_example.txt*

**borrar líneas**

**seq 5 | sed '2,4d'** = borro línea de 2 a 4

**seq 5 | sed '2d;4d**' = borro línea 2 y 4

OJO al punto coma y la coma que cambia

**seq 10 15 | sed '/14/d'** = borra las líneas que CONTIENEN “14”

**sed '/this/d' Text\_example.txt** = borrar líneas que contienen “this”

¡ = signo exlamación inverso. Para quedarme con una línea

**seq 10 15 | sed '3!d'** = me quedo con la línea 3

**TRABAJANDO CON FICHEROS COMPRIMIDOS**

**ZIP**

**zip [output] [fichero1] [fichero 2] …**

**unzip [fichero]** = para descomprimir

**zipinfo** = te dice cuantos ficheros tienes dentro

**zcat** = muesta el contenido de un fichor comprimido, pero sólo de un ARCHIVO!! si hay varios ignorará el resto. POr eso es importante saber cuántos archivos hay dentro del zip.

**zless** igual que less pero sobre el fichero comprimido

**zcat fichero.zip |** y hago lo que quiera con ese texto

**zgrep** igual que grep pero con ficheros comprimidos

**unzip -c output.zip optd\_aircraft.csv**

-c = devuelve el contenido a la pantalla descomprimido

**GZIP**

Su extensión es .gz

Siempre tiene únicamente un fichero dentro, por ello no hay que dar el nombre de la salida. Sólo enumeramos los ficheros uno detrás de otro, y borrará el original.

-k = keep, te deja el fichero original. Aunque dice el profe que normalmente si no hace falta no mantienen los originales.

***gzip*** *optd\_aircraft.csv optd\_airlines.csv optd\_por\_public.csv*

**gunzip** \*.gz = descomprime todos los ficheros de esa extensión

**BZ2**

se usa para ficheros grandes, mayores de 128mb

igual que antes tenemos:

bzip2

bzcat

bzless

**TAR**

sirve para agrupar y comprimir, o sólo agrupar.

-c = create

-z = zip (compresor que utilizamos)

-j = si elegimos algorithmo de bzip2 para comprimir

-f = fichero. Poner nombre de fichero output después de la f.

**tar -czf "optd.tar.bz2"** optd\_airlines.csv optd\_aircraft.csv optd\_por\_public.csv

para comprimir con bzip2

**tar -cjf "optd.tar.gz"** optd\_airlines.csv optd\_aircraft.csv optd\_por\_public.csv

la extensión que he puesto da igual, no influye.

siempre se usa tar tanto para comprimir como para descomprimir. Osea para descomprimir usamos extract

-x = extract

**tar -xf** optd.tar.gz

si no queremos comprimir, solo agrupar no ponemos no –z ni –j

**tar –cf** [fichero1] [fichero2] …

**SCRIPT**

Un programa escrito en Shell

Queremos un script con

2 parametros: delimitador y nombre de ficher.

Output: número y su columna

se le suele poner al archivo del script la extensión .sh (de shell)

Creamos un fichero llamado csv\_number\_column.sh y lo abrimos con otro ubunto para ir visualizando lo que metemos en él.

lo abrimos con:

nano csv\_number\_column.sh

metemos en él lo siguiente:

*#!/bin/bash*

*paste <(seq 1 10) <(head -1 optd\_aircraft.csv | tr "^" "\n")*

*head -1 optd\_aircraft.csv | tr "^" "\n" | wc -l*

La primera línea dice en donde está instalado la Shell con la que quieres que se interprete. Es por si lo ejecuta alguien que en su ordenador tiene otras shells. OJO! No es un comentario, es # + ! y va en la primera línea, no en otra!

cuando ponemos en el terminal:

bash csv\_number\_column.csv

or

./csv\_number\_column.csv

ejecuta lo de dentro

**$PATH**

variable de entorno que contiene unos directorios. Se pueden añadir directorios así:

**PATH=my\_new\_dir:$PATH** = añade mi Nuevo directorio al principio. Usamos : porque es el separador que usa la variable path.

Hacemos esto porque queremos tener un directorio con my\_scripts y que siempre estén disponibles para ejecutarlos.

**echo 'PATH=$PATH:~/my\_scripts' >> .bashrc** = metemos nuestro nuevo path en el fichero de configuración del Shell (.bashrc)

Salimos del Shell y volvemos a entrar para que el fichero de configuración se ejecute.

Ahora desde el terminal podemos llamar al comando **csv\_number\_column.sh** desde cualquier sitio!!!

**2019.11.23 COMMAND LINE**

**BACKGROUND PROCESS**

**ctrl + c** para matar un proceso.

**&** = poniendo este símbolo al final del estatement manda su ejecución al background y podemos seguir usando el Shell.

ejemplo: *$ find -type f -name "\*.txt" -exec ls -l {} \; -exec sleep 5 \;* ***&***

el proceso de arriba muestra un txt en la pantalla cada 5 segundos.

**fg** = escribiendo esto se trae el proceso al primer plano

para mandarlo al background de nuevo:

**ctrl +z** = suspende el job, lo deja parado, cuando el job está en el primer plano.

**bg** = lo devuelve al background.

¿y cuando hay varios procesos en segundo plano?

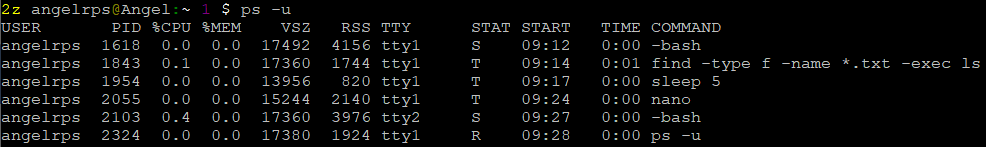
**Jobs** = te muestra los procesos que están en background, si están corriendo o suspendidos.



El número [1] o [2] es el asociado a proceso. Para matarlos suamos:

**kill %**1 = mata el proceso 1.

ps –u = muestrs procesos lanzados por mi usuario



EL número PID es el Process ID y también hace referencia al proceso. Si usamos **kill** con este número entonces **NO** ponemos %

**CSV TOOLKIT**

sirve para visualizar un csv con sus columnitas y tal desde el terminal

**pip install csvkit** = para instalarlo

**csvlook –d “^” [fichero csv] | less** = manda a less el csv con delimitador ^y lo veo en columnas

**csvlook –d “^” [fichero csv] | less –s** = con la opción –s salta a la nueva línea sólo al final de la línea del csv no del terminal, y sólo funciona con less

**csvstat –d “^” [fichero csv] | less** = muestra info de cada columna

¡¡No se puede utilizar con ficheros grandes!! porque tarde en abrirlos.

**csvcut –d “^” –c iata\_code [fichero csv]** = muestra sólo la columna iata\_code

para poner varias columnas se separan con coma SIN ESPACIO:

**csvcut –d “^” –c iata\_code,manufacturer [fichero csv]** = devuelve columnas iada\_code y manufacturer

NOTA: el delimitador por defecto que csv toolkit usa es la coma (,)

**csvcut –d “^” –c iata\_code,manufacturer [fichero csv] | csvstat** = saca estadísticas solo de las dos columnas.

En este caso no le pongo delimitador a csvstat porque por defecto es la coma, y ya viene con coma del commando csvcut anterior.

**csvgrep** = para encontrar una palabra en una columna en concreto.

**csvsql** = útil si usas sql. Yo no tengo ni idea.

csvsql –d “^” optd\_aircraft.csv

**PYTHON**

**python** = lo ponemos para empezar a trabajar en Python en el terminal

**crtl + d** or **exit()** = para salir de Python

**IPYTHON**

es una versión mejorada de Python, para facilitar su uso. Te salen las líneas nomeradas, funciona el autocomplete, etc.

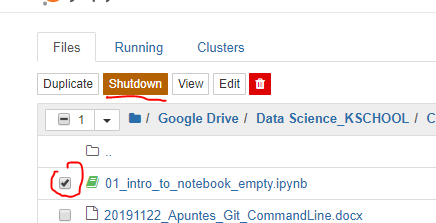
IPYTHON es sobre lo que se basan los notebooks. Los notebooks son para explorar cosas, llegar a tu comando y luego ya lanzarlo en ipython. El notebook no se usa para producción.

**JUPYTER NOTEBOOK**

se puede abrir Anaconda desde el terminal. Lo que hace es crear un servidor que va a interpretar cosas de ipython.

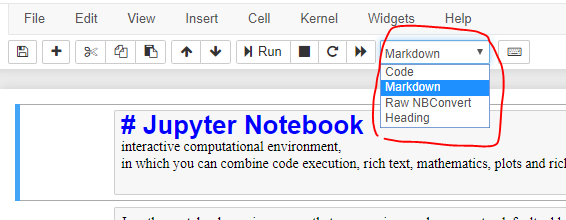
se puede cerrar el notebook desde el navegador y el servidor sigue disponible. Hasta que lo cerramos desde el terminal, el notebook sigue siempre abierto y consumiendo memoria. Es decir que cerrando la tab del navegador no se cierra un notebook completamente, la tab es sólo un visualizador.

Si abrimos un notebook aparece en verde



aunque cerremos la tab sigue abierto, para cerrarlo hay que seleccionarlo y click en shutdown.

Tipos de células:



shift + enter = ejecutar una célula

modo azul = comandos afectan a la célula

modo verde = comandos afectan al interior de la célula

Markdown: interpreta texto con formato

Raw = no interpreta nada, es decir texto sin formato

Code = lo interpreta como código Python

a = crea una célula above

b = crea una célula below

x = borra una célula

h = help

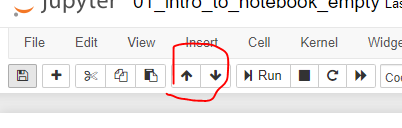
c = copiar una célula (sin ctrl)

v = pegar una célula (sin ctrl)

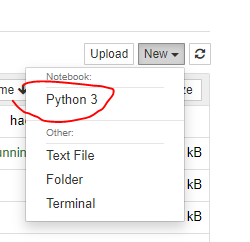
ctrl + z = sólo funciona en el interior de la célula (modo verde. No hay control z global. Cada célula tiene su propio historial.

alt + enter = ejecuta la celula y crea una nueva debajo

flechitas para mover células arriba y abajo

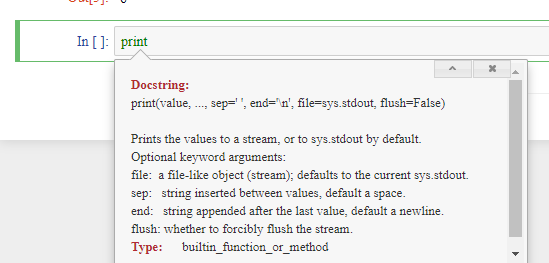


crear nuevo notebook



Cuando se ejecuta una celda sólo te muestra la última línea.

**shift + tab** = pulsado sobre una función te sale la ayuda



type(variable) = me dice el tipo de variable (integer, string, etc.)

el intelisense sobre una variable se activa con tab

Se puede crear una tupla sin usar los paréntesis:

c = 5,6 = es una tupla

Un set es como un diccionario pero solo con keys

Se crear como un diccionario sin poner los values.

**isinstance()** – check i fan object is of a particular type

**hasattr**(c, "capitalize") = para preguntar sit u variable tiene un atributo en concreto

dic (objeto) = muestra todos sus atributos, incluidos los privados! Los privados no salen en el autocomplete. Para eso sirve dir() para ver todos los que realmente podemos utilizar.

IMPORTANTE

para abrir un notebook de anaconda desde Linux pongo

**jupyter-notebook** en la línea de comandos

me devolverá una dirección que copio y pego en Chrome

