# **Clase 1: Trabajar con Git en U­buntu**

Comandos más útiles por orden de utilización:

* **mkdir**: Crea un directorio
* **pwd**: te dice en que directorio estas
* **ls**: te da todos los directorios que carpetas y archivos tienes en el directorio actual
* **cd:** te abre un directorio o archivo
* **±**: te indica que estas dentro de un repo (si esta en verde quiere decir que no hay cambios desde la versión original, por defecto la palabra que sale es ‘’master’’, si está en rojo quiere decir que se han hecho cambios que no están guardados en github, si está en amarillo quiere decir que esta guardado el commit pero en local). Cuando añades/eliminas una línea pasa a ser (+1/-1)(+2/-1)… según añadas o elimines.
* **git clone <link>:** descarga el git que haya en ­­el link
* **cat:** muestra el interior del documento
* **echo:** muestra el texto que escribas después (echo Hola que tal : Hola que tal)
* **echo ….. > file** : SUSTITUYE el contenido (…) el archivo (file)
* **echo ….. >> file** : AÑADE el contenido (…) el archivo (file)
* **git status:** te dice el estado de la ‘’rama’’
* **git pull:** descarga y UNE los cambios hechos
* **git add <file>:** añade los cambios que se han hecho para que se suban cuando termines.
* **git reset:** te devuelve a la fase anterior
* **git config:** te permite decir el usuario que escribe los cambios:
  + **git config—global user.name “……..”:** cada espacio separa una acción. Global te lo cambia para todos los archivos
  + **git config—global user.email “……..”:** cada espacio separa una acción. Global te lo cambia para todos los archivos
* **git commit:** te permite añadir los commits que HAYAS AÑADIDO:
  + **-**m: te permite escribir un mensaje
* **git push:** sincroniza con github y sube el repo que has cambiado
* **git checkout—**file: quita los cambios y pasa de rojo a verde
* **git reset HEAD <file>:** quita los últimos cambios que hayas añadido con ‘’git add’’.
  + **git reset HEAD^:** quita el último commit completo. Puedes hacerlo más veces para quitar tantos commits como quieras. MUY ÚTIL PARA ERRORES
* **git log:** te da el log de los cambios
* **nano <file>:** abre un editor de texto para modificar el archivo a mano:
  + cuando pone ^ significa Ctrl
* **git init <nombre>:** Crea un repositorio nuevo

Proceso:

Pull (clone la primera vez) -> haces los cambios -> git add <file> -> git commit -> git push

Para ir hacia atrás:

* git reset HEAD <file> : quita lo última que hayas ‘’git add’’
* git checkout --<file>: quita los cambios que hayas hecho en el original

# 

# **Clase 2: Línea de comandos para Data Science**

## **Variables y comandos útiles**

La primera palabra es siempre el comando, y lo que viene después son parámetros que incluyes. Si va entre “\_\_\_” se toma como un solo parámetro.

Para descargar archivos se puede usar **curl y wget.** Las principales diferencias son:

* Main difference is that wget is used to download things while curl is used to test communication by using a specific protocol.
* wget's major strong side compared to curl is its ability to download recursively.
* wget is command line only. There's no lib or anything, but curl's features are powered by libcurl.
* curl supports FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS, FILE, POP3, IMAP, SMTP, RTMP and RTSP. wget supports HTTP, HTTPS and FTP.
* curl builds and runs on more platforms than wget.
* wget is released under a free software copyleft license (the GNU GPL). curl is released under a free software permissive license (a MIT derivate).
* curl offers upload and sending capabilities. wget only offers plain HTTP POST support.

Las variables se definen igual que en Python. Estas variables y parámetros quedan guardados según el Shell que estés utilizando.

Comandos útiles:

* Si pones **“$...”** te interpreta la variable que tu pongas después de $
  + $SHELL: te dice que interpretador estás utilizando
  + $TERM te dice el terminal que utilizas
  + $<var>. : te devuelve el valor de la variable <var>
* **whoami**: te dice con qué usuario estás trabajando
* **printenv**: imprime todos los environments
* **cat -n .bashrc**: lee el número de líneas del fichero
* **cp**: te permite copiar archivos o directorios
* **type**: comando que te dice el tipo de ‘’comando’’ y como viene instalado y de donde viene.
* **which**: te dice que es lo que va a hacer cuando uses un comando (archivo que va a ejecutar, por ejemplo)
* **whereis**: te dice donde está
* **man:** permite buscar la información de un comando. Se busca con **/**
* **wc <file>**: te dice el número de líneas, palabras y tamaño del archivo:
  + **wc -l:** devuelve solo el número o de líneas
  + **w**c -**w:** devuelve las palabras
  + **wc -c:** devuelve los bytes
* **<comando + lo que quieras> > <file>:** incluye todo lo que pongas en la acción del comando en el fichero <file>. Permite DIRECCIONAR la información.
* **find:** es importante cuando busques un directorio poner primero un punto, por ejemplo find ./Shell para que entienda que empieza en ese directorio.
  + **-name “<name\*>”:** devuelve lo que encuentre que empiece con ese nombre gracias al \*
  + **-iname:** igual que el anterior pero sin distinguir entre mayúsculas y minúsculas.
  + **-type <d><f>**: te devuelve los directorios ‘’d’’. Para archivos sería ‘’f’’
  + -**maxdepth <número>:** busca en una cantidad finita de niveles por debajo
  + -**mindepth <n>:** al revés que el anterior
  + **-perm <n>:** busca archivos con unos permisos en concreto
  + **! :** niega el siguiente parámetro
  + **-empty:** busca ficheros vacíos
  + **-size <n><+n> <-n>**: busca tamaños según pongas + o –. Si no pones nada busca ese exacto
  + **-mmin <n><-n>:** busca según hace cuanto que has modificado algo
* **-exec:** ejecuta el comando que tu le pongas después.
  + Ejemplo: **find . -maxdepth 5 -type f -iname "Text\_File\*" -exec echo "Found it" \;** ES NECESARIO PONER AL FINAL \; YA QUE EL FIND BUSCA UN ; AL FINAL DE CADA EJECUCIÓN Y PARA QUE NO SEA INTERPRETADO LO PONEMOS ASÍ.
  + Ejemplo 2: **find . -maxdepth 5 -type f -iname "Text\_File\*" -exec ls -l {} \;** Aquí aplica igual que el .format de Python, donde están las {} pondrá la información que ha recibido (en este caso el ls -l de lo que vaya encontrando). Si no lo pusiésemos aparecería el ls -l de lo que tenemos en la carpeta actual tantas veces como resultados encuentre. Se pueden concatenar ‘’exec’’.
  + Se pueden ejecutar varios comandos a la vez de la siguiente forma:
    - Exec sh: ejecuta una Shell
      * -c: da los parámetros de la nueva Shell
        + “comandos”

Ejemplo:

**find . -maxdepth 5 -type f -iname "Text\_File\*" -exec ls -l {} \; -exec sh -c "ls -s {} | wc” \;**

El fichero de configuración de bash se puede modificar.

## **Interpretación de variables**

Hay una **diferencia entre usar comillas dobles y simples**: Cuando utilizas comillas simples NO interpreta lo que pongas dentro, lo imprime o usa tal cual, si usas comillas dobles SI que lo interpreta y lo sustituye:

* ’ ’ - Single quotes: no tocar el texto
* “ ” - Double quotes: Perform shell variable expansion
* ` ` - evaluate and replace (command substitution)
  + Be careful!!! ` IS NOT EQUAL TO ‘
  + $(command) not equal to $command !!!!

Si dejas las **comillas ABIERTAS** te permite seguir escribiendo en el pront:

* Echo “world (ENTER y se habré una línea que permite seguir escribiendo hasta que cierres las comillas).

Lo anterior también aparece si se usa **\**:

* Echo “world” \ is mine (y aparecerá todo seguido)

Si utilizas el **\ con algo que hay después** le estas diciendo que lo que hay después no te lo interprete, simplemente te lo imprima y ya esta. Hay casos especiales:

* Primer se debe utilizar el parámetro **-e**:
  + \t= tabula (EN BASH NO FUNCIONA, SOLO EN ZSH)
  + \n = crea una nueva línea (EN BASH NO FUNCIONA, SOLO EN ZSH)

Si quieres separar dos comandos en la misma línea hay que utilizar **punto y coma ;**. Si quieres proteger el ; y que te salga como tal debes poner \;

Ficheros especiales:

* cat /etc/os-release : te da el sistema
* cat /proc/cpuinfo : te da el hardware

## **Ejemplo para trabajar y movernos por los archivos de un repo**

Con cd nos movemos por el directorio. Si usamos tabulador después de ‘’cd ‘’ nos aparecerán las opciones disponibles.

Para abrir varios ficheros seguidos:

* **cat -n <file> <file>. Además agrega el número de línea que es**

Si queremos que abra varios archivos con alguna variable usamos \*:

* **cat \*.txt:** abrirá todos los ficheros que terminen con .txt

Cuando utilizas cat .. (dos puntos) estás indicando que estás un directorio por encima.

* **cat ../../** : estás indicando que estas DOS directorios por encima

Para leer ciertas líneas:

* **cat -n <file> | head -n <número>:** abre un archivo, le pone el número de línea gracias a ‘’-n’’, con el | ejecutará el comando de la derecha una vez abierto el archivo, con head leera las primeras líneas, al poner -n le estas diciendo que ahí va un número, y el <número> puede ser:
  + **negativo:** cogera las últimas X líneas
  + **positivo:** cogerá las primeras X lneas

Puedes dar el directorio completo a mano.

Si queremos dar un path desde nuestra carpeta de usuario ponemos poner directamente:

* / ~

Si queremos volver al directorio anterior:

* **cd - O cd ../**

Tabulando podemos autocompletar y buscar directorios:

* **cd** ./ **:** si autocompletas te busca los directorios
* **cd . :** si autocompletas te busca los directorios escondidos
* **cd :** vuelve al directorio home/joeldela

Para trabajar dentro de un fichero podemos usar el comando **less**:

* una vez pones **less <file> -N: (el -N pone el número de cada línea, si usas -S te pone la línea entera aunque sea más ancho que la pantalla)** 
  + **pulsar g:** se mueve al principio del archivo
  + **/<word>:** busca la palabra DESDE DONDE ESTAS HACIA DELANTE
  + **?<word>:** busca la palabra DESDE DONDE ESTAS HACIA DETRÁS
    - **pulsar n:** enseña el siguiente resultado de búsqueda
    - **pulsar N:** enseña el anterior resultado de búsqueda
  + **/^<word>:** busca la palabra que EMPIEZA con <Word>
  + **/<word>$:** busca la palabra que termina con <Word>

## **Comandos para propiedades de archivos**

Comandos útiles para ls:

* **ls -l** : te dice los archivos que hay y sus datos. También se podría utilizar LL (en minúscula)
* **ls -s** : te da el tamaño
* **ls -a**. te proporciona los archivos escondidos
* **ls -ls** : junta ls -l y ls -scd
* **ls -lt**: te ordena los ficheros por orden de tiempo
* **ls -ltr**: ordena los ficheros por orden de tiempo inverso (el más viejo primero)
* **ls -lSr**: ordena los ficheros por tamaño (con la r lo hace en orden inverso, primero el más pequeño)

Un archivo en un color especial quiere decir que es un archivo que se puede ejecutar

Para darle el permiso de ejecutar a un usuario (a mi mismo en este caso) sería:

* **chmod u+x <file>:**
  + u: es el usuario (u de user para mi mismo)
  + g: para grupos
  + o: others si quieres darselo al resto
  + x: ejecutar
  + Para escribir sería w por ejemplo
  + +: se usa + porque le estas DANDO permisos, para quitarlos sería –

Para dar los permisos se utilizan códigos binarios. 111 en binario suma 7. Por lo que según queramos dar el permiso a cada uno de los parámetros (1 o 0 para ejecutar, 1 o 0 para escribir, 1 o 0 para leer, y esto para cada uno de los usuarios/grupos/others), como se le darían las tres cosas a cada ‘’persona’’, se pondría chmod 777 y se le dan todos los permisos a todos. Si pones 757 le das a ‘’u’’ todos los permisos, a ‘’g’’ le das dos permisos y uno no, y a ‘’o’’ le das todos los permisos.

## **Modificar directorios**

Se pueden crear directorios de diferentes formas:

* **mkdir <dir>** : crea el directorio <dir>
* **mkdir <dir> <dir> :** crea dos directorios al mismo nivel
* **mkdir -**p <dir/dir> : crea un directorio dentro de otro

Para crear un fichero se utiliza el comando **touch**.

Para copiar ficheros:

* **cp <fich a copiar> <nombre fichero nuevo>**
* **cp -r <direc a copiar> <nombre para el direc copiado>**
* **cp <file> <file> <dir> :** copia todos los ficheros al directorio
* **cp -p …… :** copia el archivo con los permisos

Para cambiar el nombre de los archivos y mover:

* **mv <nombre a cambiar> <nuevo nombre>**
* **PARA MOVER Y RENOMBRAR A LA VEZ ES COMO MOVER EL ARCHIVO A UN DIRECTORIO QUE TENGA EL NOMBRE NUEVO QUE QUEREMOS DARLE:**
  + Ejemplo: **mv text.txt sub1/text2.txt** nos va a mover el archivo text.txt a la carpeta sub2 con el nombre nuevo de text2.txt

Para eliminar:

* **rm**
* **rm -r :** permite borrar directorios
* **rm -fr** : borra todo sin preguntar nada

## **Atajos de teclado:**

* **ctrl + r**: te permite buscar el último comando que has ejecutado que contengas las letras que tu escribas
* **ctrl + k:** borra lo que haya delante del cursor
* **ctrl + l**: te lleva a la última línea
* **| :** manda lo que haya a la izquierda y ejecuta el comando que haya la derecha. Es una forma de concatenar comandos

# **Clase 3: Comandos avanzados en Shell**

## **Modificar archivos**

* seq <inicio> <intervalo(opcional)> <final>: da una secuencia de valores

El comando **uniq** nos devuelve los valores únicos de un fichero. **Solamente mira las líneas que van seguidas**.

* + Utilizando el parámetro uniq **-c** nos cuenta cuantas veces se repite
  + Utilizando el parámetro uniq -**d** nos imprime solamente los que están duplicados.

Para que sea útil habría que ordenar el fichero. Para ello se utiliza siempre el comando **uniq** con el comando **sort**:

* Ejemplo. sort uniq\_example.txt | uniq
  + Con el parámetro **-n** ordena de forma numérica, sino por defecto lo hace de forma ‘’alfabética’’
  + Parámetro **-r**: orden inverso
    - sort -nr uniq\_example.txt uniq\_example.txt uniq\_example.txt: esto mostraría en pantalla 3 veces el mismo fichero pero ordenado numéricamente
  + Parámetro **-u:** muestra los valores únicos del fichero. La desventaja es que no puede contar el número de veces que aparece ni aparecen los duplicados.
  + Para este comando el ‘’delimitador’’ que habrá dentro del archivo se especifica con el comando **-t**. Es el único en el que el parámetro es con una t, en el resto es con una d.
    - Ejemplo: sort -t "^" -k6r,6 optd\_aircraft.csv | head : este ordenaría el fichero en base a la columna 6 hacia delante en sentido inverso.
    - Ejemplo 2: sort -t "^" -k2r,2 -k7r,7 optd\_aircraft.csv | head : en este caso nos permitiría ordenar primero según una columna y luego según otra.
    - EJERICICO: sort -n 20lines.txt | uniq -d -c | sort -k1r | head -n 1 : con este comando podríamos buscar la línea que más veces se repite en un fichero e imprime esta línea.

El comando **cut:** permite cortar un cacho de tabla

* + cut -d "^" -f 1-3,5 optd\_aircraft.csv | head : en estre caso estamos cortando parte del archivo, las columnas 1 a 3 y la 5. (el **-d** es el delimitador).
  + cut -d "^" -f 1-3,5 --output-delimiter=";" optd\_aircraft.csv | head : en este caso estamos cortando igual que antes pero además estamos cambiando el separador a ‘’;’’.

El comando **tr:**  te permite cambiar un símbolo por otro. Es importante que debe ser DEL MISMO TAMAÑO EL QUE VAS A CAMBIAR Y EL QUE CAMBIAS. Si no son del mismo tamaño, va a rellenar los huecos con el mismo patrón. Sirve mucho para los delimitadores:

* + <file> | tr “<elemento a cambiar>” “<elemento que queremos>” (puede llevar o no las comillas, depende)
  + El parámetro **-s** viene de ‘’squeeze’ por lo que ‘’elimina’’ los caracteres que tu introduzcas cuando se encuentran seguidos. A continuación, si pones estos códigos entre comillas, permite efectuar el parámetro en:
    - [:alnum:] letras y números
    - [:alpha:] letras
    - [:blank:] espacios en blanco
    - [:digit:] números
    - [:lower:] todas las letras en minúsucula
    - [:upper:] letras en mayúsculas
  + El parámetro **-d** elimina los caracteres que elijas.
  + El parámetro **-cd** busca e imprime los caracteres que pongas

El comando **paste** te permite unir en columnas distintas ficheros. Junta la primera fila del primer fichero con la primera línea del segundo fichero y así sucesivamente. Definimos FILAS. Es muy útil por ejemplo para definir la columna de un archivo de texto: creas un archivo con una secuencia de números y luego haces paste <archivo con la secuencia> <archivo a numerar>:

* El parámetro **-d** permite introducir el separador que queramos ponerle
* El parámetro **-s** permite copiarlo horizontalmente. Es decir, defines COLUMNAS.
* Puedes poner como parámetro solamente **–** y con ello le indicas en el número de columnas que quieres que lo divida. x

Un caso especial es cuando quieres enviar el contenido de un fichero a un comando. Para ello se utiliza **<comando> <(comando <file>). ES IMPORTANTE QUE NO SE PONGA UN ESPACIO DESPUÉS DE <.** Con ello lo que haces es que devuelves el contenido al comando al comando inicial.

* Ejemplo: paste <(seq 15) <(cat Text\_example.txt)
* Ejemplo: paste <(cat numbers) <(cat Text\_example.txt)

***Mirar ejercicios de las diapositivas.***

* Cambiar el delimitador: cat optd\_por\_public.csv | tr "^" "," > optd\_por\_public\_comma.csv
* Contar columnas: head -1 optd\_por\_public\_comma.csv | tr "," "\n" | wc -l
* Imprimir la columna con su número correspondiente: paste <(seq 46) <(head -1 optd\_por\_public\_comma.csv | tr "," "\n")

El comando **grep** nos permite buscar una línea que contenga una palabra en concreto:

* Grep <palabra> <file> : nos devuelve LA LÍNEA en la que aparece
* El parámetro **-n** nos da el número de línea en el que está.
* El parámetro **-v** nos da las líneas QUE NO CONTIENEN esa palabra
* El parámetro **-c** nos cuenta las líneas que tienen esa palabra.
* El parámetro **-iE** permite que no distinga entre mayúsculas y minúsculas.
* El parámetro **-A** **<número>** imprime el número de líneas que vienen después de la línea en la que se encuentra la palabra y la línea de la palabra
* El parámetro **-B** **<número>** imprime el número de líneas que vienen antes de la línea en la que se encuentra la palabra y la línea de la palabra
* Para hacer ambos a la vez se usa **-C**.
* Si queremos buscar una linea que EMPIECE por una palabra se pone ^: “^<palabra>”
* Para buscar una palabra que CONTENGA un valor variable (7x7 siendo x un numero cualquiera) se puede usar un conjunto de valores:
  + Grep -E 7[0-9]7

***Ejercicio de las diapositivas: buscar las aerolíneas que empiecen por ‘’aero’’ en una columna concreta:***

* head -1 optd\_airlines.csv | tr "^" "\n" | wc -l : contamos las columnas
* paste <(seq 14) <(head -1 optd\_airlines.csv | tr "^" "\n") : vemos el número de columna
* cut -d "^" -f 8 optd\_airlines.csv > columna8 : metemos la columna 8 en un archivo
* grep -i -E "^aero" columna8 | wc -l : contamos las aerolíneas que empiezan por ‘’aero’’ sin tener en cuenta mayúsculas y minúsculas
  + Resumido en una línea: cut -d "^" -f 8 optd\_airlines.csvls | grep -i -E "^aero" | wc -l

El comando **sed** permite transformar y editar. Primero se define que se quiere hacer y después se define el delimitador:

* Sed s/day/night/ : la primera s es de ‘’substitute’’, y lo siguiente son los delimitadores. Se puede usar cualquier cosa siempre que sean el mismo. En este caso solo lo cambiaría UNA vez.
* sed s/day/night/g : añadiendo esta **g** permitiría cambiar la palabra ‘’day’’ a ‘’night’’ en TODO el documento.
* El parámetro **-i** permite hacer los cambios **DENTRO** del fichero
* El parámetro **-n** permite **leer** líneas:
  + ‘2p;4p’ mira la 2 y la 4
  + ‘2,4p’ mira de la 2 a la 4
  + Si no lo pones duplica las líneas que has puesto
  + Como se observa, al poner ‘p’ después de los números **LEES** las líneas.
  + Si pones **!p** lo que hace es que lee las líneas que NO tengan la palabra: /line/!p
* Si pones ‘d’ en lugar de ‘p’ **BORRAS** la línea
  + Un ejemplo de borrar una línea: seq 5 | sed '2d'
* Si queremos borrar una línea que contenga X carácter:
  + Sed ‘/X/d’ : aunque esto no es recomendable.

## **Ficheros comprimidos**

### zip

El comando **zip**  permite meter en un .zip los archivos que quieras:

* **zip** <nombre.zip> <archivo1> <archivo2>….

El comando **unzip** descomprime los archivos:

* El parámetro **-l** nos da la información de lo que hay dentro.
* El parámetro **-c** saca el archivo que tu elijas a la carpeta en la que elijas.

Para lectura de archivos zip se puede emplear:

* zcat: solo lee el primer archivo que hay, el resto los ignora.
* zless: Permite leer en un editor de texto el archivo zip
* zgrep: permite buscar en el zip

### Gz

Se trata de un formato que guarda solo 1 fichero. El comando a emplear es **gzip**. Creará un fichero gz por cada archivo y lo sustituirá.

* **gzip** <archivo1> <archivo2>…..

El comando para volver a tener los archivos en formato original es **gunzip**

### Bz2

El comando para comprimir en este formato es **bzip2.** Creará un fichero bzip2 por cada archivo y lo sustituirá. Se utiliza para archivos que son muy grandes**:**

* **bzip2** <archivo1> <archivo2>…..
* **bzcat :** permite leer el archivo
* **bzless :** igual
* **bzgrep:** permite buscar en archivos bzip2

El comando para volver al archivo original es el **bunzip2**

Si se quiere meter varios archivos se puede hacer:

* bzip2 \*.txt
* bunzip2 \*.txt

### Tar

Permite juntar varios ficheros en uno. El comando será **tar**:

* Los parámetros a usar son **-cvf** para crear un fichero ‘’tar’’.
  + **tar -cvf <file.tar> <archivos…..>**
* Los parámetros a usar son **-czvf** para crear un fichero ‘’zip’’.
  + **tar -czvf <file.tar.zip> <archivos…..>**
* Los parámetros a usar son **-cjvf** para crear un fichero ‘bz2”.
  + **tar -cjvf <file.tar.bz2> <archivos…..>**
* El parámetro **-xf** permite descomprimir el archivo

## **Crear un script**

Una vez que tienes escritos el script, para inciarlo se empleará el directorio del bash para iniciar su ejecución.

Siempre habrá que poner **#!/bin/bash**

**IMPORTANTE: Para las variables, para asegurar que se ejecuten, poner siempre ${VARIABLE}. VA ENTRE LLAVES.**

Para añadir estos scripts a los comandos se tiene que añadir la carpeta en la que se encuentran dichos comandos a PATH:

* PATH=$PATH: ~/<nombre de la carpeta>

# **Clase 5: Comandos avanzados y Python**

# Ejecutar comandos en segundo plano

Para ejecutar un proceso en el segundo plano habrá que poner al final del comando el carácter **&**. De esta forma se queda ejecutando en segundo plano.

* El comando **bg** lleva al background lo que estés ejecutando
* El comando **fg** lleva al frontground lo último que se este ejecutando en segundo plano

Otro editor de texto es **gedit**. Permite abrir un editor de texto distinto al nano.

Con el comando **jobs** se muestran los comandos que estás ejecutando. Este comando solo vale para la Shell que estés ejecutando, si tienes dos abiertas no te mira lo de las 2..

Con el atajo **ctrl+z** suspendes un comando.

Con el comando **ps** obtienes la información de los procesos y el usuario que los está ejecutando.

* Ps -U <nombre de usuario> | grep gedit : te buscaría el número de proceso del comando ‘’gedit’’

## **CSVkit**

El primer comando de CSVkit es **csvlook.** Con este comando se puede ver el csv. NO SE PUEDE utilizar sobre ficheros GRANDES.

* El parámetro **-d** te permite meter el delimitador para que te lo separe bien
* El parámetro **-l** te da el número de línea

Si quieres leer bien una tabla puedes usar el comando **less** y luego ponerle el parámetro **-S** para que te lo separe bien en filas.

Otro comando útil es **csvstat** que nos da información acerca del fichero (estadísticas). Si quieres elegir una columna basta con usar el parámetro **-c <nombre de la columna>**.

El comando **csvcut** funciona igual que el comando **cut** normal pero para archivos csv

* El comando **-c** nos permite seleccionar las columnas que queramos:
  + Si queremos números: 2-4,6 (selecciona de la 2 a la 4 y la 6)
  + Si queremos podemos dar el nombre de la columna.

También están **csvgrep, csvsort…**

El comando más útil es **csvsql**. Este comando nos da la información necesaria de cada columna para SQL para crear una tabla.

* El parámetro **-d** vuelve a ser el delimitador.
* El parámetro **-i <tipo de SQL>** permite que te lo saque en el formato que tu quieras.