Nosotros nos moveremos en lenguajes de programación interpretados principalmente. En los compilados hay que traducir en el lenguaje de las máquinas, y esto es más difícil y menos intuitivo.

Javascript (para páginas web) y HTML son pseudo-lenguajes de programación.

Codesignal: para aprender a programar con juegos.

Cachés: para agilizar el procesamiento, cuando ya sabe qué vas a pedir. Pero hay veces que eso fallará.

Fichero de log es un diario. Las apps escriben TODAS las operaciones que realizan en un log.

Analogía de comensal, camarero y cocina para entender qué es una API.

Encriptación de doble lado (RSA): clave pública y privada, que son dos caras de una misma moneda, una llave y una cerradura. Lo que se pone en las páginas web es la pública; la privada no se la da a nadie.

**SSH**: lenguaje de servidor a servidor.

**FTP**: ficheros.

SMTP: enviar emails.

HTTP: web (hypertext transfer protocol).

Nosotros nos quedaremos en la capa de aplicación. En concreto, en los dos primeros protocolos.

Una IP es una dirección, la del ordenador, pero tiene múltiples entradas-salidas o puertos. El puerto ha de ser el mismo. El estándar es el 80. Será necesario el mismo protocolo (lenguaje) y mismo puerto (sitio) para entendernos (camarero y yo).

Unidades de procesamiento: CPU (regresión lineal, p.e.); GPU o gráfico (para detectar si una imagen es un perro o un gato, p.e.); TPU (TensorFlow y entrenamiento de modelos); y FPGA.

Sistemas operativos: Mac OS, Windows y Linux, principalmente.

**Datalización**, que las empresas pasen a ser dirigidas por los datos.

Archivos: un fichero de texto es simplemente un documento que contiene información, puede ser un CSV un fichero de texto, pero no tiene porqué. Un CSV tiene siempre la estructura como en la imagen: valor, coma, valor, coma, etc. El “encoding” (UTF-8) es importante para que sepan en qué tipo de idioma le estás hablando, qué significa ese código. Si me envías un texto y hay caracteres extraños que no son letras eso sería un encoding diferente y no nos entenderíamos.

Lector plano es necesario, como un bloc de notas.

CheatSheet: una serie de comandos de chuleta. Hay de Python y de otros, como UNIX.

Descargar todo el software que pone ahí, en especial Slack (para comunicarse).

GIT: como el portfolio de lo que has hecho.

MS-DOS: Microsoft/Windows; UNIX: Mac y Linux.

**FUNDAMENTOS UNIX**: Tiene un sistema de archivos jerárquico, rígido, no como Windows, y todo en ficheros de texto, no como Windows (que hay de todo). Todos los devices son también ficheros de texto (un teclado, p.e.). Es multi-tarea. Es LINUX.

Pwd: dónde estoy.

Whoami: quién soy.

Sudo su: para ser root o super usuario. Si quiero hacer algo con superusuario tengo que poner antes de lo que sea “sudo”.

Exit: para volver a ser cadoca.

Hay varios usuarios: el tuyo y el super usuario, que puede hacer de todo.

Cd /: cambiar directorio al fichero raíz.

Ls: lista de ficheros.

Ls – l:

Por columnas y en orden es: permisos primero.

(Si somos root o no: sudo) + función + argumentos

Ejemplos:

Ls (función) –l (argumentos)

Ls (función) –lt (argumentos)

Ls (función) –lrt (argumentos)

Ls –lh (es lista pero larga y en lenguaje comprensible para humanos)

Para saber qué hace cada uno mirar la CheatSheet.

Cd . no te lleva a ningún sitio

Cd .. navega a la carpeta de arriba

Cd – volver atrás

Cd /home para que te lleve a ese directorio en concreto

Cd : te lleva al fichero anterior, he subido un nivel

Cd nombre carpeta para volver a la carpeta donde estaba antes de subir el nivel

Cd / te lleva al fichero raíz

Mkdir + nombre carpeta es para crear una carpeta donde estás

Mkdir + ruta + nombre carpeta es para crear una carpeta en la ruta esa. Por ejemplo, mkdir /tmp/test-dir crea en /tmp el directorio test-dir

Mkdir –p para crear directorios padres que falten hasta el que quieres crear

Touch es crear un fichero

Si pongo touch ruta/nombre.txt creo el archivo nombre.txt dentro de la ruta

Help x te dice qué es x

Man x te dice qué es x, es igual que help pero diferente

Cp origen destino es copiar una cosa de un sitio a otro

Cp –r si es un directorio lo que estoy copiando (se copian los archivos a los que llama el directorio, que a fin de cuentas es solo otro archivo que llama a otros archivos que se incluyen en él, entonces tiene que ir copiando y volver a copiar cada vez todos los que incluye (por eso se llama recursivo a la r).

Mv es mover de origen a destino también, pero no creo copia, solo arrastro, es como hacer una copia y un borrado. Mv sirve también para renombrar. Si es un archivo con espacios entonces se pone entre comitas.

Rm es borrar. Si pones \* es para incluir varios con algo en común o todos en general

Solo si eres sudo puedes borrar cosas creadas con sudo

Para borrar algo tienes que estar dentro del escalón jerárquico que corresponda, sino no te dejará borrar, habrá que moverse (cd) y luego ya hacer rm.

Rmdir es para borrar directorio

Rm **–rf**: borrar un directorio, si no está vacío. Recursivo y a la fuerza.

Cat para ver qué hay dentro de un fichero (no directorio)

Se puede poner alias ll=’ls –lh’ por ejemplo para que al escribir ll ya salga todo eso y no tengas que escribirlo todo cada vez.

El comando history es para saber tu historial de comandos utilizados.

Si le das al tabulador te rellena la palabra y si le das a flechita para arriba te pone el último comando utilizado y así sucesivamente.

Hay que respetar siempre la jerarquía, no se puede ir a un sitio pasándote otros niveles. Para crear no se puede estar en / sino en /home/cadoca

Lo que no tiene extensión suelen ser carpetas (directorios) y si tienen extensión son ficheros (files).

Echo “hola mundo” > X sustituye contenido de X

Echo “hola mundo” >> X concatena, añade en X

En diferentes ramas dos ficheros o carpetas se pueden llamar igual; en la misma rama no.

Cd $HOME para ir a /home/cadoca (Users/carlosdonosocabero en Mac), cuando estés perdido, que es donde interesa estar (también simplemente poner cd y ya) para crear cosas y modificar y tal.

En la terminal de Mac “home” es “Users” (creo).

*Ejercicio 1:*

Cd /

Cd Ls-l

Para abrir archivos ejecutables (programas como los que acaban en .sh) darle a ./nombre y se abrirá.

Permisos:

Propietario, grupo y generales, en orden.

R (lectura), W (escritura) y X (ejecución): cada uno de los tres grupos puede tener uno, dos o tres de estas funciones.

Chmod Calculator para obtener un número en función de los permisos que quieres tener

Chmod + número + archivo es para elegir los permisos a tu gusto

Chmod 777 –r es lo más usado y general para todos poder hacer todo

-r significa recursivo

df –h para saber información sobre la unidad de disco

Instalar apps siempre con sudo (sudo apt install …)

Cat vuelca todo el contenido de un fichero en la pantalla.

Date enseña la fecha y la hora.

*Ejercicio 2:*

1. Ls –l, esos ficheros pertenecen a mi usuario.
2. Mkdir yo + ls –l
3. chmod 700 yo (con eso leo solo yo)
4. chmod 711 ella (primero crear ella con touch)
5. ls –lt /etc y miras el último
6. df –h
7. du –h . es para saber cuánto ocupa cada fichero o directorio

Poner todos los nombres de directorios y tal en minúscula y sin espacios.

Un conjunto de líneas de código o instrucciones que se ejecutan para un propósito en concreto es un **Script**.

#!/bin/bash

mkdir PEDRO

chmod 777 PEDRO

touch hola.txt PEDRO/.

Vi + fichero/archivo, para rellenarlo, te sale un texto plano (sales con ESC + :wq si quieres guardar y sino pues con :q)

Nano es otro diferente pero como vi, y para salir seria Ctrl X + Yes + Enter

Visual Studio Code (hay enlance en el pdf para acceder a shell bash online)

**Si, mientras, durante**: IMPORTANTE, es la base de la programación: if, while, for.

*Ejemplo* en CodingGround (SEC) o Bash:

variable="hola"

if [ $variable = "hola" ]; then

echo "adios"

fi

if [ $var = "hola" ]; then

echo "adios"

else

echo "hola"

fi

else quiere decir “sino”

El $ es para que te de el valor del carácter, para diferenciarlo del carácter en sí.

GitKraken, Typora y Git

Google Cloud Pricing Calculator

**FUNDAMENTOS GIT**

Controlador de versiones es para saber el histórico de un documento, para rastrear su pasado. Para saber si es un controlador de versiones o no, hay que preguntarse las 5 W’s. Es un software que nos permite versionar documentos. Y hay que añadir siempre un comentario explicando un poco la situación (why), que responde a las 5 W’s. Trabajan con ficheros de texto plano (Sharepoint).

En el **centralizado** hay un repositorio central y luego tu local (ordenador personal, p.e.) y desde local con “commit” se pasa al repositorio cada vez que cambias tu documento. Si quiero traer un documento del repositorio al local hago un “pull” o un “update”. Hay tantas versiones en repositorio como veces se haya modificado el documento, con independencia del contenido. Importante siempre hacer “pull” primero, porque si haces una versión v2 y ya hay una v2 (misma línea temporal) en repositorio entonces te pedirá hacer un pull y al hacerlo te borrará tu v2 y te la cambiará por el v2 del repositorio y ahí ya podrás cambiarlo.

*Ejercicio*:

L0: COMMIT A 123

L1: CREA B ABC

L0: CREA C ABC

L2: PULL

L2: MODIFICA A 1234

L0: MODIFICA A 143

L0: COMMIT A

L1: MODIFICA B ABD

L1: COMMIT B

L2: CREA B ABC

L2: COMMIT B

Para pasar de v1 a v2 tiene que ser el mismo documento y cambiarlo.

Cuando haces un pull te llevas todo del respositorio, pero *solo las últimas versiones*.

No se pueden contradecir las versiones de un mismo tipo. No sabes si tienes conflicto hasta que haces commit en el centralizado, porque el control se hace en remoto centralizado.

En el **descentralizado** o *distributed version control* (como GIT) aparece un nuevo comando, el “push”. En nuestro local podemos trabajar sin conexión a Internet. Se pasa de mi work a mi repositorio local con commit (sin conexión a Internet) y luego hago push para pasar a repositorio central. Puedo traerme las cosas a local para que estén seguras antes de cargármelas como pasaba antes si había conflicto; hay aislamiento del repositorio central; y además tengo un histórico en mi local. Esas son las 3 ventajas. Y el update es como el push pero al revés. Aquí los conflictos se resuelven en local, no en repositorio central.

*Ejercicio*:

L1: CREA A 123

L2: CREA B ABC

L2: PUSH: no pasa nada

L1: COMMIT

L2: UPDATE: no pasa nada

L2: MODIFICA B ABD: es v1 aún hasta que haga commit, entonces sería v2

L1: PULL: no pasa nada

L1: UPDATE: no pasa nada

L2: PUSH: no pasa nada

Si hago un push y tengo varias versiones se sube todo, *no únicamente las últimas versiones*. Si haces update es igual, te bajas todo de repositorio a local. Pero si luego haces pull entonces a tu work sí que te baja solo la última versión. Así conseguimos tener los históricos completos en local.

Otro comando que es “add” para seleccionar los archivos concretos que quiero subir, porque con commit se sube todo. Se hace add y luego commit\*\*.

Brancheado, Master (rama madre a la cual no se puede hacer commit), pull request (para saltar de una rama a otra hay que pedir permiso a otro desarrollador), feature branch. El último que sube a la rama tiene que incluir lo que han subido los otros de otras ramas a la rama principal, p.e.

GitHub: es una empresa que ofrece repositorios de GIT (es un software de controlador de versiones). Markdown (es el lenguaje) CheatSheet y StackEdit.

GitKraken:

Para crear o modificar, el comando es add (stage)\*\*.

WIP: work in progress (working directory or work).

Para poner una imagen es ![Alt Text](url), luego se da a preview changes y a commit para que se guarde. Todo esto dentro del README.md + editar.

<https://www.tutorialspoint.com/unix/unix-getting-started.htm>

<https://www.learnshell.org>

<https://www.katacoda.com/courses/git>