01

# Conceptos generales

Javascript con React y Node JS

fundación esplai
ciudadanía comprometida

# Networking + Internet

# Conceptos generales de networking

**Red: Conjunto de ordenadores (u otros dispositivos) que se "comunican".**

- red local
- red global

¿Qué significa que se comunican?

- Se hacen preguntas y respuestas
- Se pueden enviar paquetes de información

**¿Como se entienden?**

- Mediante protocolos

**Todos los equipos de la red son "hosts", pudiendo actuar como:**

- **Clientes**: principalmente hacen preguntas
- **Servidores**: principalmente dan respuestas

# ¿Qué es Internet?

Internet es una red pública y global de ordenadores* que están interconectados mediante el protocolo de Internet (Internet Protocol).

(wikipedia agosto 2016)

(* y otros aparatos)

# Los servicios de Internet

Lo que "vemos" de Internet son sus utilidades o aplicaciones. Las hacen posibles los servicios.

Los servicios los ofrecen los servidores especializados repartidos por la red.

**Servicios**

- HTTP HyperText Transfer Protocol (WWW World Wide Web)
- FTP File Transfer Protocol
- SMTP Simple Mail Transfer Protocol
- Altres: P2P, NNTP, VoIP, SSH/Telnet...

**Todos comparten un protocolo de base: TCP/IP**

# El Protocolo de Internet: TCP/IP

Transfer Control Protocol / Internet Protocol

**Protocolo**: conjunto de normas que regulan la forma como se preparan y envían los paquetes de información, permitiendo a los ordenadores de una red entenderse.

Las direcciones que identifican los ordenadores en una red TCP / IP se denominan **direcciones IP.**

**La dirección IP es la matrícula que el router (o el ISP) asigna a un ordenador o dispositivo en el momento en que se conecta a Internet. Es imprescindible para poder formar parte de la red.**

- Los servidores tienen direcciones estáticas, vinculadas a una URL a través del servicio DNS
- Los clientes suelen tener direcciones dinámicas, pueden cambiar a cada nueva conexión.

Una IP es un conjunto de 4 bytes o 32 bits. Por lo tanto son 4 valores entre 1 y 255:

- 172.16.0.1
- 10101100.00010000.00000000.00000001

# ¿Qué significa la dirección IP?

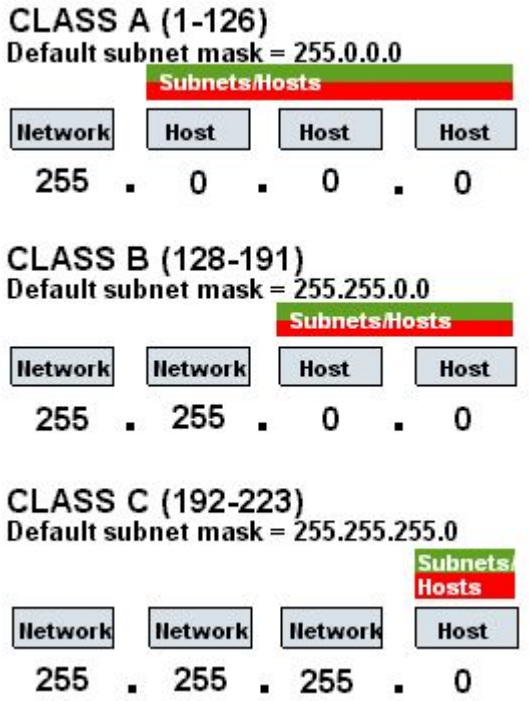
El primer byte identifica la clase de la red:

- Clase A: gobiernos
- Clase B: ISPs, corporaciones
- Clase C: "Pequeñas" redes

Redes privadas (rangos de IP reservados):

- 10.0.0.0 - 10.255.255.255 (16,777,216)
- 172.16.0.0 - 172.31.255.255 (1,048,576)
- 192.168.0.0 - 192.168.255.255 (65,536)

Direcciones especiales: 127.0.0.1 (...)



# IPs i URLs

Los ordenadores se conocen entre sí por sus IPs. Así, un ordenador entiende perfectamente estas direcciones:

- c3po@199.181.132.250
- http://93.184.221.131:80

Mientras que los humanos vemos más claro esto otro:

- c3po@starwars.com
- http://www.pokemongo.com

**Estas IP's "traducidas" son las URL (Uniform Resource Locator)**

La parte de la URL que identifica al servidor o host al que nos referimos es el **dominio**. En los casos anteriores: starwars.com o pokemongo.com

URI, URL, URN...

- URI Uniform Resource Identifier. Nombre genérico de un recurso, nuestro nombre por ejemplo.
- URL Uniform Resource Locator. Nombre de un recurso, incluida la forma de llegar a él (http, ftp...)
- URN Uniform Resource Name. Nombre único de un recurso, ej) código ISBN de un libro.

URL válidas:

- http://example.com/mypage.html
- ftp://example.com/download.zip
- mailto:user@example.com
- file:///home/user/file.txt
- tel:1-888-555-5555
- http://example.com/resource?foo=bar#fragment

# Dominios i subdominios

**Un dominio** es un nombre simple que utilizamos para identificar un servidor, que normalmente ofrecerá contenidos web.

- **mediamarkt.cat**

Contiene dos partes. La parte derecha es el **Top Level Domain**, que pot ser de unos tipos predeterminados, por ejemplo:

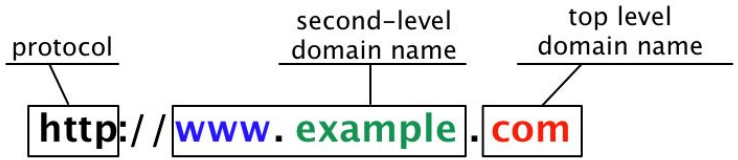
- No restringidos: .com, .info, .net, .org
- Restringidos: .biz, .name, .pro
- Comerciales: .barcelona, .pizza, .taxi...
- Geográficos: .fr, .pt, .uk, .es, .cat...

**La parte izquierda es el Second Level Domain**, que en realidad es el nombre principal del dominio. Puede ser cualquier nombre "libre" de 3 o más caracteres (sin espacios ni signos "raros").

**Subdominios:** se sitúan a la izquierda del nombre de dominio principal:

- tiendas.mediamarkt.es

La correlació entre noms de domini, servidors i adreces IP es gestiona a través del DNS.



# Acceso a Internet: tecnologías

Tecnologías de conexión **fija** o cableada**:**

- **ADSL:** Dsl asimétrica, utiliza cables de cobre convencionales. Hasta 9Mb down, 640 kb up.
- **Cable:** cable coaxial, el mismo que ofrece la tv por cable. Poco extendido aquí. 2x DSL.
- **Fibra óptica:** Cable de fibra que transmite datos como impulsos de luz. Máximo caudal, aprox 300 Mb / 30 Mb

Tecnologías de conexión **móvil** o sin cable:

- WIFI: poco alcance, para redes locales. Hasta 100 Mbps.
- 3G, HSDPA, 4G: Amplio alcance, aunque coste elevado. El plan de datos mide caudal y volumen mensual. Velocidades reales de 5 a 20 Mbps.

**OJO!** se habla de **velocidad**, pero en realidad es **caudal**! (Volumen de datos / tiempo)

# Caudal ("velocidad") y volumen de datos

| Nombre | Símbolo | Bytes |
|---|---|---|
| kilobyte | KB | 10^3 (o 2^10) |
| megabyte | MB | 10^6 (o 2^20) |
| gigabyte | GB | 10^9 (o 2^30) |
| terabyte | TB | 10^12 (o 2^40) |
| petabyte | PB | 10^15 (o 2^50) |
| exabyte | EB | 10^18 (o 2^60) |
| zettabyte | ZB | 10^21 (o 2^70) |
| yottabyte | YB | 10^24 (o 2^80) |

1 Byte = 8 bits

La "velocidad" o caudal se mide normalmente en múltiplos del bit, y sus múltiplos se representan con la "b" minúscula (Kbps, Mbps) y el sufijo ps (por segundo) o la expresión Mb/s, Kb/s ...

Los volúmenes de datos son la medida más habitual, se representan con la "B" mayúscula según la tabla de la izquierda.

# El Software

# Software

El software es aquello "no físico" del ordenador:

- Los programas o instrucciones que deberá ejecutar la unidad de proceso. También le llamamos la "lógica".
- La información que debe manejar el ordenador, del tipo que sea

Todo el software, sean programas / aplicaciones o información, está codificado en **sistema binario** y guardado en dispositivos de memoria. Aunque no es "físico", el software ocupa un espacio en esta memoria (sea memoria de 1º o 2º nivel).

Centrándonos en el ámbito de programas o "instrucciones", dividimos el software en dos grandes grupos:

- Los sistemas operativos
- Las aplicaciones

# Sistema operativo

Programa o conjunto de programas de un sistema informático que **gestiona los recursos de hardware y provee servicios** a los programas de aplicación (software), **ejecutándose en modo privilegiado** respecto de los restantes. *Wikipedia*.

# Software: el Sistema Operativo

- Ofrece una base de compatibilidad a las aplicaciones
- Gestiona la comunicación de éstas con el hardware
- Proporciona los elementos básicos de la IU (Interfaz de usuario)
- Incorpora los protocolos de comunicación con otros ordenadores (redes)

# Servicios y Aplicaciones

Desde el punto de vista del SO todo los que se "ejecuta" son “procesos” o “tareas”. Como usuarios, distinguimos:

- **Servicios**. Ejecutados en "background", sin interfaz de usuario o muy reducida. Dan "servicio" a otros programas.
  - Bases de datos, Web, Email, Streaming, etc.
  - En linux se suelen llamar "daemons" o "demonios"
- **Aplicaciones**. Interactúan con el usuario y por tanto disponen de interfaz.
  - Ofimática, CAD, Lenguajes de programación, Vídeo, Fotografía, Comunicaciones, etc.

# Software: las Aplicaciones

Aportan la funcionalidad al ordenador, permitiéndole realizar tareas concretas, especializarse. Algunos tipos de aplicaciones:

- Herramientas y lenguajes de programación
  - Permiten construir otras aplicaciones
- Aplicaciones de servidor ( "daemons")
- Aplicaciones de comunicación (clientes)
- Utilidades, antivirus...
- Ofimática, diseño gráfico...
- Aplicaciones multimedia: edición de video, fotografía...
- Aplicaciones de entretenimiento, juegos
- Aplicaciones de compartición peer-to-peer
- ...

# Software: las Aplicaciones

Si son específicas para un sistema operativo llamamos "**nativas**", y sólo funcionarán en el SO para el que han sido creadas (aunque puedan tener versiones para otros SO).

Las aplicaciones **multiplataforma** son aquellas que pueden ejecutarse en varios sistemas operativos, normalmente apoyándose en una capa intermedia que sí es específica del SO. Es el caso de las aplicaciones WEB o muchas aplicaciones mobile basadas en HTML5 / CSS / Javascript. También de JAVA o aplicaciones .NET

En el caso de las aplicaciones WEB la "capa de contabilidad" multiplataforma es el navegador (Chrome, Firefox, Safari ...)

# Desarrollo de software "Programación"

# Lenguajes de programación

Estructuras sintácticas y semánticas utilizadas para controlar el comportamiento de una máquina.

Tipos de lenguajes:

- Interpretados / compilados
  - Pros/Cons
- Alto nivel / bajo nivel
- Tipados / no tipados

Algoritmo, script

- Secuencial, ordenado

Algunos ejemplos:

- Interpretados: PHP, Python, Ruby, Javascript
- Compilados: C++, Java, C#

# Lenguajes de programación

**Estructuras sintácticas y semánticas utilizadas para controlar el comportamiento de una máquina.**

Escribiremos los programas en un lenguaje de programación concreto (Javascript, Java, C#, C++, PHP, Ruby, Scala, ...)


Algoritmo, programa, script

- Conjunto de instrucciones
- Secuencial, ordenado

# Paradigma de programación

Propuesta tecnológica aceptada por una comunidad de programadores como una forma de hacer frente a los problemas que plantea el desarrollo de software.

**Programación imperativa o estructurada.**
La ejecución sigue una secuencia explícita de órdenes que modifican el estado (de la aplicación: variables, datos...).
Los algoritmos repetitivos se encapsulan en funciones.
Ej. Basic, Pascal, C

**Programación orientada a objetos.**
Se utilizan "objetos" que interactúan entre sí.
Tienen su propio estado "interno" e interfaces públicas (encapsulamiento).
Pueden ser:

- Basados en clases: los objetos adquieren su comportamiento y estado según la clase a la que pertenecen. Ej. Java, Python, C#
- Basados en prototipos: toman el comportamiento a partir de un objeto prototipo. Ej. Javascript

# Código fuente y código ejecutable

- Lo que nosotros escribimos como programadores es el “código fuente”
- Podemos escribir programas mediante editor de texto…
- Pero existen programas especializados llamados IDE (Integrated Development Environment).

- Ni el SO ni la CPU pueden entender el código fuente, necesitan **código ejecutable**.

- Para crear código ejecutable a partir de c. fuente disponemos de los programas llamados **“compiladores” o “intérpretes”**

# Compiladores...

- Los compiladores leen los archivos de código fuente y los convierten en otros archivos de código ejecutable, normalmente específico para cada SO.
- Este código ejecutable (los .exe de Windows o .app de OSX) no interactúa directamente con el Hardware sino que se apoya en el SO para realizar llamadas. Por tanto sólo será ejecutable en el SO para el que lo hayamos compilado

# Compiladores /2

Por ello, para poder ejecutar nuestro programa "compilado" en distintos SO (Mac OS, Windows, Linux, ...) necesitaremos:

- Un compilador del lenguaje usado para cada SO
- O bien “algo” que se ponga por medio entre el SO y nuestro programa. Una capa más.
    - JVM / Java Virtual Machine o la plataforma .NET son ejemplos de esta "capa" añadida que se sitúa entre el SO y nuestro código.

# Intérpretes

Los intérpretes son aplicaciones (o servicios) que traducen el código fuente en código ejecutable en tiempo real, en el mismo momento en que es necesario.

Javascript es un lenguaje interpretado. El propio navegador (o NodeJS) son los encargados de traducir el código fuente a ejecutable.

Otros lenguajes interpretados son PHP, Ruby o Python.

# Compiladores vs intérpretes

COMPILADORES

- No requieren instalar un intérprete en el cliente.
- El código fuente original queda protegido.
- Normalmente la conversión es más optimizada, no requiere la inmediatez.
- El compilado analiza todo el código, no la parte que se ejecuta en ese momento. Son menos propensos a errores.

INTÉRPRETES

- El código suele ser multiplataforma.
- El proceso de desarrollo suele se más simple.
- Tenemos a la vista todo el código, librerías... Es más difícil esconder un virus.
- El intérprete puede adaptarse mejor a las características de la máquina cliente.

# Persistencia: Bases de Datos

Los programas se "ejecutan" en la memoria del ordenador. Recopilan, procesan y muestran datos. Pero una vez terminan su función, salvo en contadas ocasiones, estos datos deben guardarse: persistir.

Aplicaciones sencillas pueden guardar los datos en archivos propios, sea en modo local o remoto. Pero en la mayoría de los casos esta tarea se delega en aplicaciones especializadas: las bases de datos.

Una Base de Datos o BDD es una aplicación que se ejecuta en segundo plano (servicio), habitualmente en un servidor (aunque puede estar en la misma máquina cliente).

Su misión es dar persistencia a los datos que manejan las aplicaciones.

Las más conocidas son las BDD de tipo SQL, siglas del lenguaje que utilizan para comunicarse con otras aplicaciones.

# Bases de datos

Aplicaciones "especializadas" en almacenar información. Normalmente residirán en un servidor.

Nuestras aplicaciones "conectan" con las BDD:

- De forma directa, mediante un enlace JDBC, por ejemplo.
- A través de una API REST o similar

En cualquier caso, al crear una aplicación **delegamos** en una BDD la gestión de la información, y nos centramos en su tratamiento y lógica de negocio.

**SQL**, relacionales o estructuradas

- Los datos residen en tablas relacionadas mediante índices o claves
- Rápidas y eficientes, pero requieren estructurar los datos
- Ej: MySQL, SQL Server

**NoSQL**

- Adaptadas a los nuevos formatos de la información (Objetos, JSON)
- Ej: MongoDB, DynamoDB

# Editores de código Integrated Development Environment

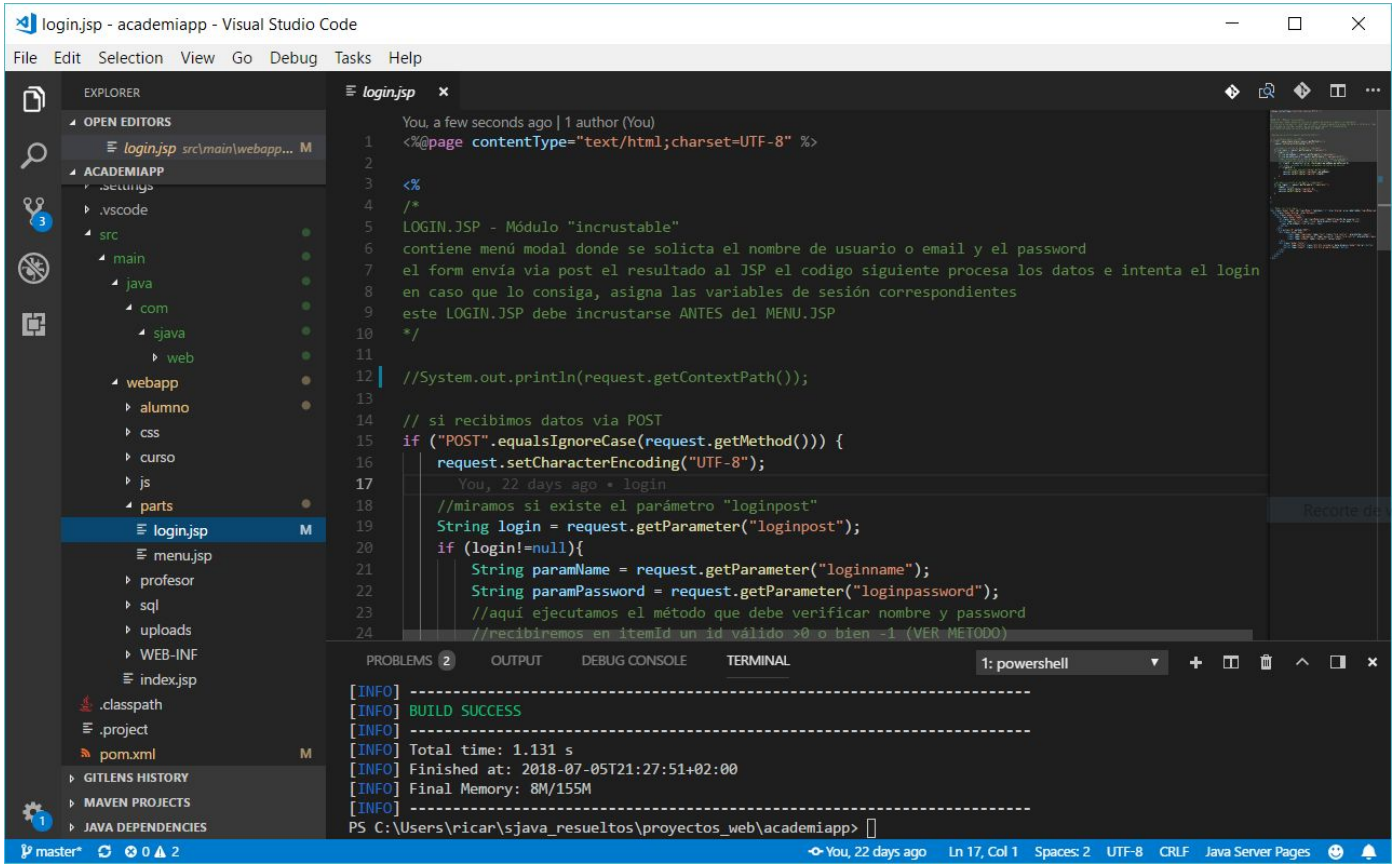
# Editores de texto/código e IDEs

Podemos crear código con Notepad, pero utilizar una aplicación especializada aporta muchas ventajas:

- Asistencia en la sintaxis
- Autocompletado de clases y métodos, importación de packages
- Integración con control de versiones (GIT)
- Detección de errores en tiempo de escritura y tiempo de ejecución (debug)
- Terminal integrado, FTP integrado, acceso a bases de datos...
- etc.

**En este curso utilizaremos Visual Studio Code (o VS Code)**

Tal vez no sea el IDE más potente, pero su facilidad de uso permite concentrarse en el código y facilita la comprensión del proceso de creación de aplicaciones, a menudo oculto en otros IDEs.


login.jsp - academiapp - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
login.jsp src\main\webapp... M
ACADEMIAPP
.vscode
src
main
java
com
sjava
web
webapp
alumno
css
curso
js
parts
login.jsp M
menu.jsp
profesor
sql
uploads
WEB-INF
index.jsp
.classpath
.project
pom.xml M
GITLENS HISTORY
MAVEN PROJECTS
JAVA DEPENDENCIES
login.jsp
You, a few seconds ago | 1 author (You)
<%@page contentType="text/html;charset=UTF-8" %>
<%
/*
LOGIN.JSP - Módulo "incrustable"
contiene menú modal donde se solicta el nombre de usuario o email y el password
el form envía via post el resultado al JSP el codigo siguiente procesa los datos e intenta el login
en caso que lo consiga, asigna las variables de sesión correspondientes
este LOGIN.JSP debe incrustarse ANTES del MENU.JSP
*/
//System.out.println(request.getContextPath());
// si recibimos datos via POST
if ("POST".equalsIgnoreCase(request.getMethod())) {
request.setCharacterEncoding("UTF-8");
You, 22 days ago • login
//miramos si existe el parámetro "loginpost"
String login = request.getParameter("loginpost");
if (login!=null){
String paramName = request.getParameter("loginname");
String paramPassword = request.getParameter("loginpassword");
//aquí ejecutamos el método que debe verificar nombre y password
//recibiremos en itemId un id válido >0 o bien -1 (VER METODO)
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
1: powershell
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.131 s
[INFO] Finished at: 2018-07-05T21:27:51+02:00
[INFO] Final Memory: 8M/155M
[INFO] ------------------------------------------------------------------------
PS C:\Users\ricar\sjava_resueltos\proyectos_web\academiapp>
master*
0 2
You, 22 days ago
Ln 17, Col 1
Spaces: 2
UTF-8
CRLF
Java Server Pages

# GIT: Control de versiones

# Observaciones preliminares

**Un proyecto de programación:**

- Está compuesto por múltiples archivos, cientos o miles fácilmente.
- Es un trabajo en equipo. Pueden intervenir decenas de programadores. O más.
- Los códigos se interrelacionan, especialmente en OOP. Es muy importante hacer un seguimiento de los cambios para "trazar" posibles errores.
- El código fuente de una aplicación puede ser el principal valor de una empresa. Asegurar su integridad es esencial.
- Las copias de seguridad diarias son ineficientes. Localizar un código antiguo borrado puede ser una tarea más complicada que reescribirlo.

# ¿Qué es un sistema de control de versiones?

Un SCV es una aplicación que nos permite llevar la "contabilidad" de nuestro código, estableciendo puntos de control que nos permiten recuperar código anterior en cualquier momento.

Otra gran utilidad de los SCV es que facilitan el trabajo en equipo, permitiendo que varias personas trabajen en un mismo proyecto.

Además, herramientas como GIT se han convertido de facto en un sistema ampliamente utilizado para la distribución de software.

Existen numerosas aplicaciones de este tipo: Subversion, CVS, Perforce... GIT (pronunciado "guit") es la más popular por ser probablemente la mejor y sin duda la más utilizada al ser de código abierto.

# Instalación de GIT

El primer paso consiste en instalar GIT, para ello descargamos la versión que corresponda a nuestro SO desde la web:

https://git-scm.com/downloads

Esta descarga instalará el "motor" de GIT en nuestro ordenador, y podremos utilizar todas sus funciones desde la línea de comandos.

Al menos una vez debemos "firmar" nuestro git, ahora es un buen momento:

```
git config --global user.email "ejemplo@ghjrl.com"
git config --global user.name "Mi Nombre"
```

# Origen de GIT

En 2005 la comunidad de desarrollo de Linux decide desarrollar un sistema abierto de control de versiones, tras tener problemas con el software propietario Bitkeeper. Se establecen las siguientes preferencias:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido (no dependiente de un servidor)
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

# Cómo guarda los datos GIT

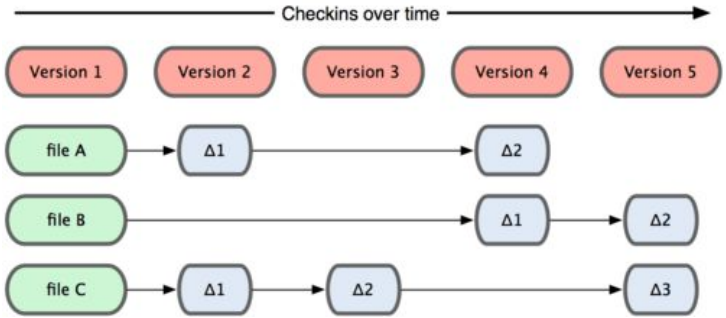
Otros sistemas de control de versiones guardan los datos como modificaciones incrementales sobre un archivo inicial.

GIT guarda "fotografías" de cada estadio del proyecto, con todos los archivos implicados (aunque no repite los no modificados).

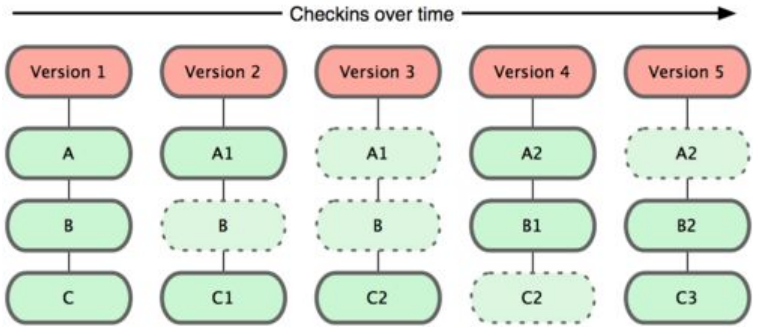
Todos los datos se guardan en modo local, en una base de datos dentro de una carpeta oculta (de nombre .git), aunque pueden replicarse en remoto. Esto proporciona mucha velocidad aunque puede ocupar mucho espacio en disco.

Esta base de datos se denomina normalmente **"repositorio"**

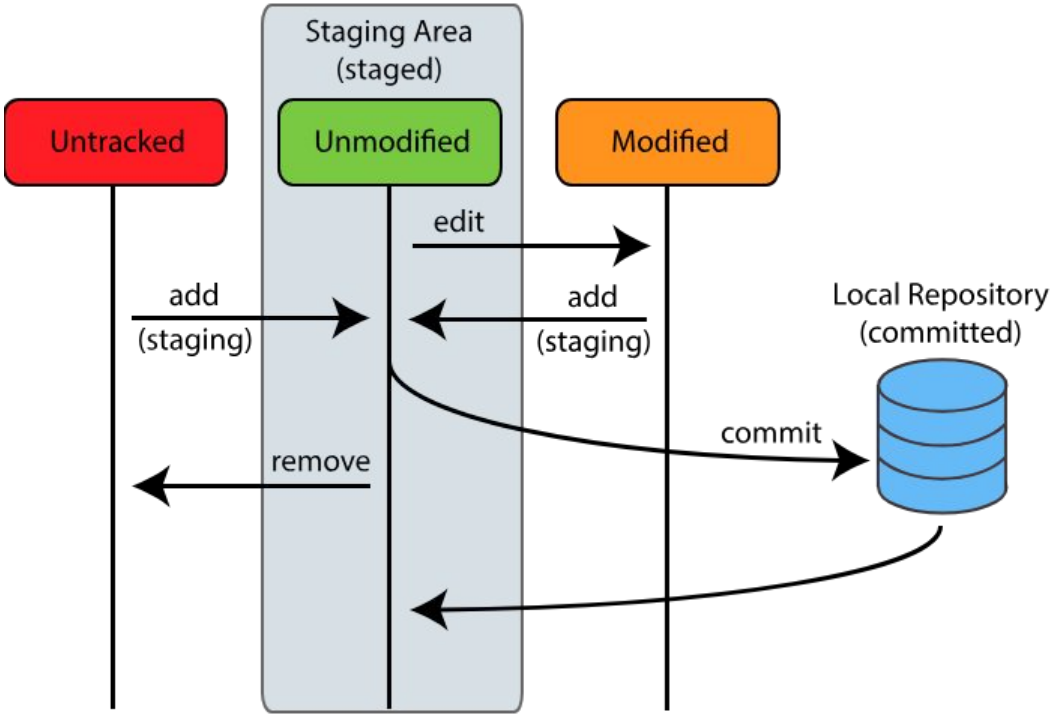
Sistema incremental



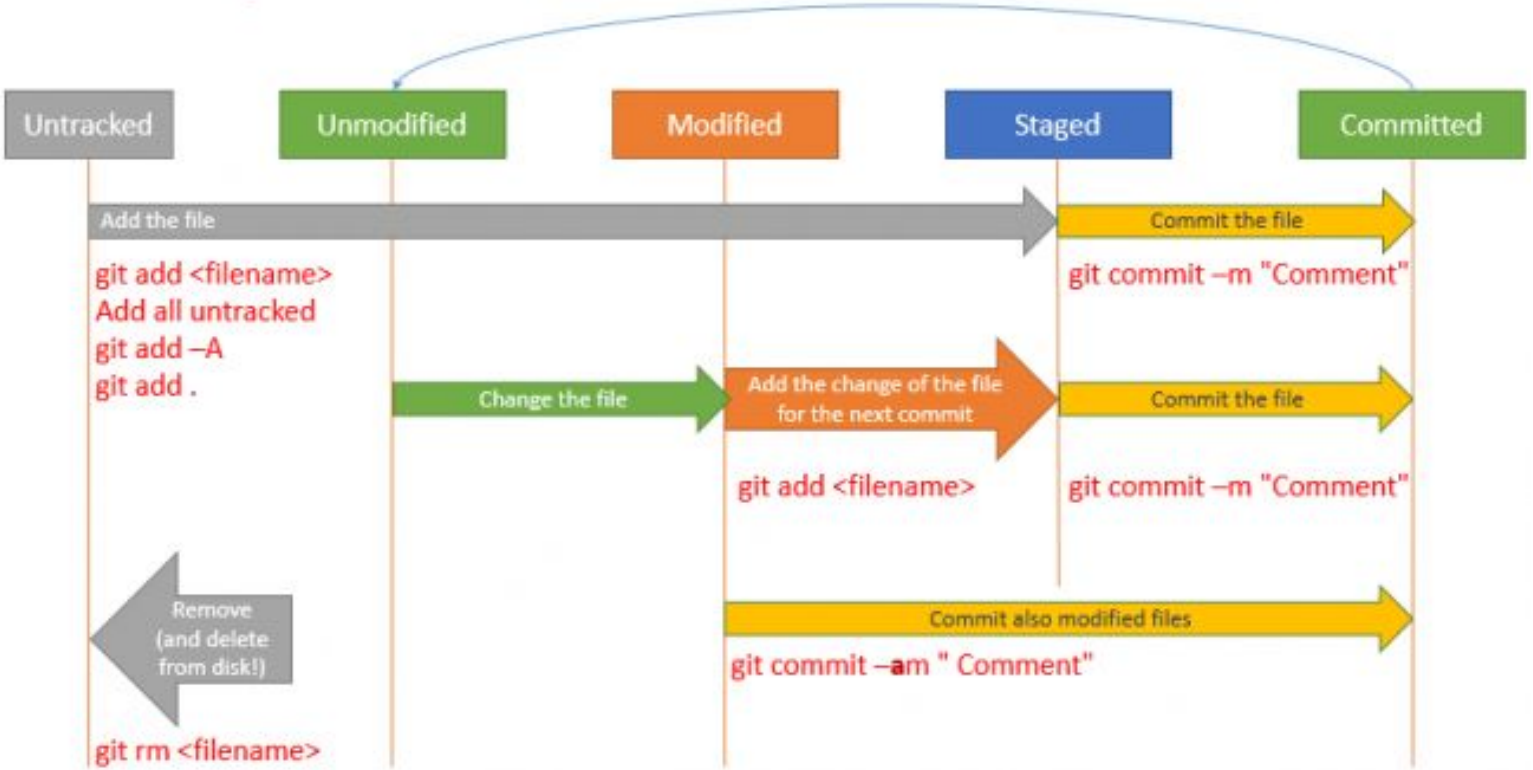

GIT: Sistema "snapshots"



# GIT "File status Lifecycle"



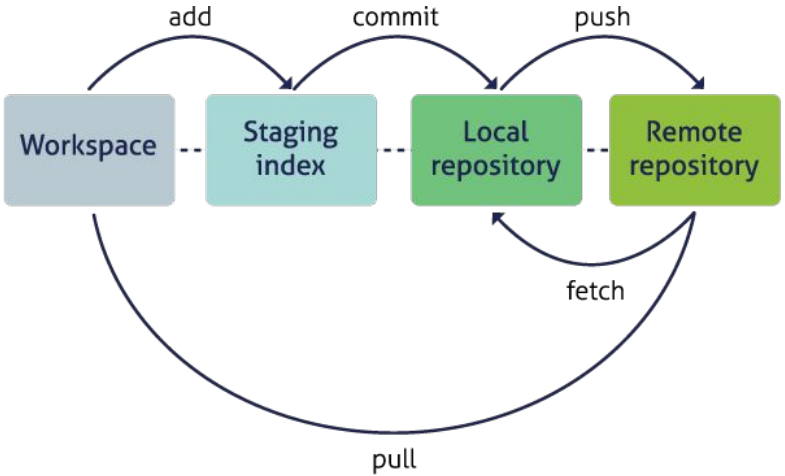
Check status with „git status“



# Repositorios remotos

Opcionalmente los datos se pueden "subir" a un repositorio online tal como GitHub o BitBucket. Esto aporta una seguridad añadida y facilita el trabajo en equipo.

Subir datos se denomina "Push", recuperarlos "Pull". Hacer una copia en nuestro ordenador de un repositorio remoto se denomina "Clone" (clonar).



# Ejercicio

- Instalación de GIT
- Creación de cuenta en bitbucket
- Creación de repositorio remoto en bitbucket: ejemplo
- Clonado a local
- Apertura con visual code, añadimos archivos
- Utilización de GIT en terminal integrado:
- git add, commit, push
- Verificación de repositorio remoto
- Ramas