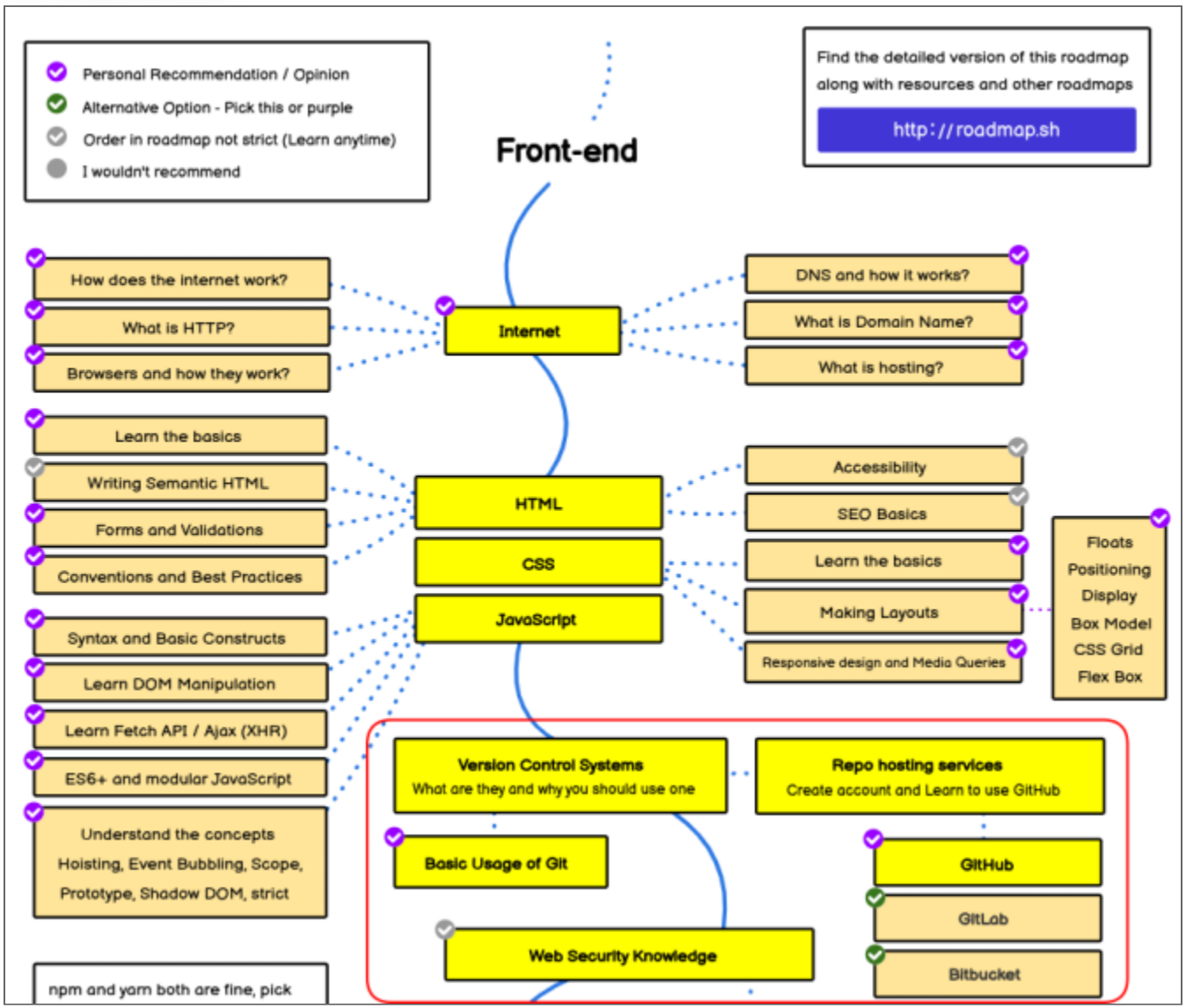
PROYECTO DE SOFTWARE

Cursada 2022

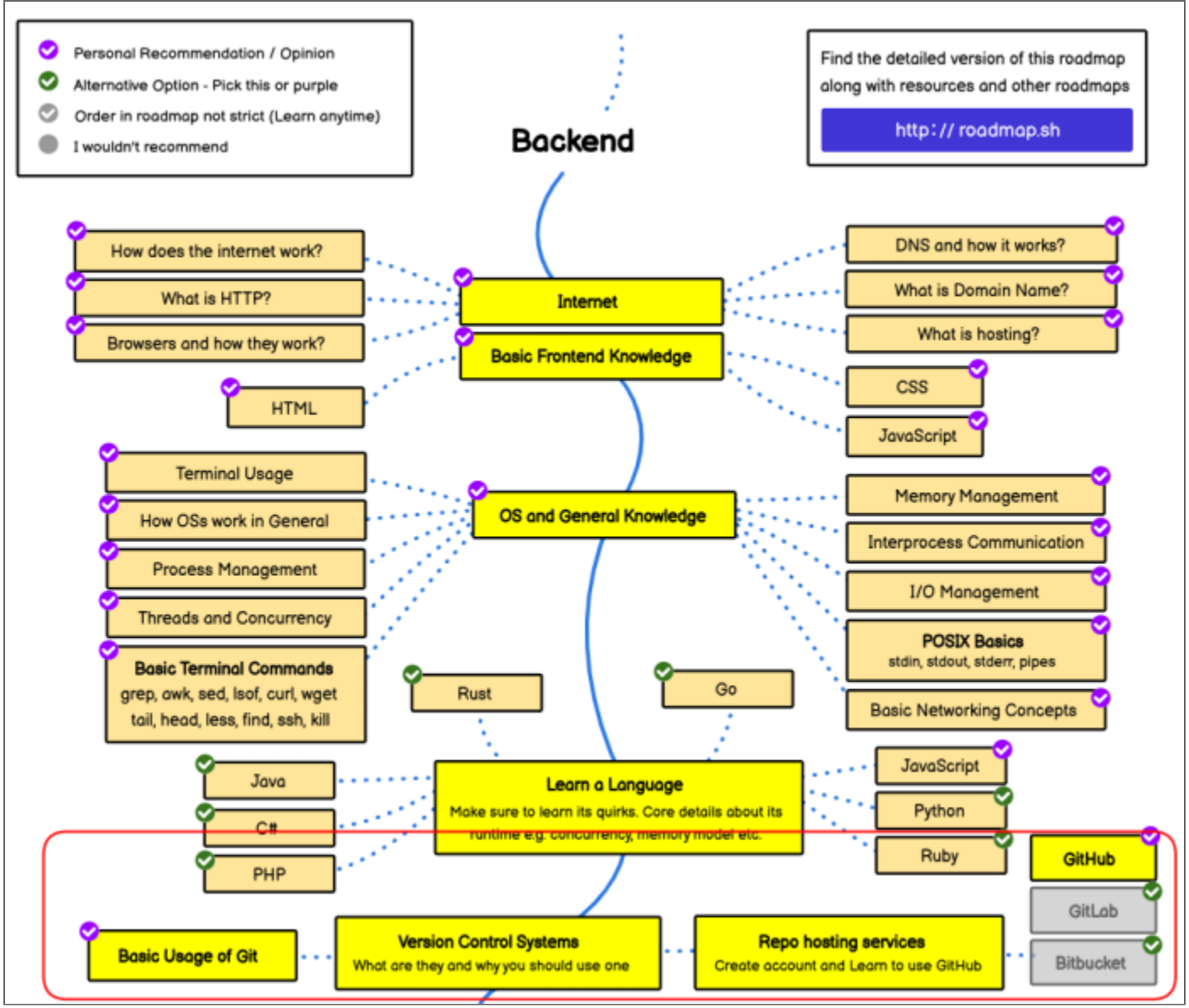
SIN IMPORTAR EL CAMINO ELEGIDO

Roadmaps del desarrollo

SI FRONTEND


Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Front-end
Find the detailed version of this roadmap along with resources and other roadmaps
http://roadmap.sh
How does the internet work?
What is HTTP?
Browsers and how they work?
Internet
DNS and how it works?
What is Domain Name?
What is hosting?
Learn the basics
Writing Semantic HTML
Forms and Validations
Conventions and Best Practices
HTML
CSS
JavaScript
Accessibility
SEO Basics
Learn the basics
Making Layouts
Responsive design and Media Queries
Floats
Positioning
Display
Box Model
CSS Grid
Flex Box
Syntax and Basic Constructs
Learn DOM Manipulation
Learn Fetch API / Ajax (XHR)
ES6+ and modular JavaScript
Understand the concepts
Hoisting, Event Bubbling, Scope,
Prototype, Shadow DOM, strict
Version Control Systems
What are they and why you should use one
Repo hosting services
Create account and Learn to use GitHub
Basic Usage of Git
GitHub
GitLab
Bitbucket
Web Security Knowledge
npm and yarn both are fine, pick

0 BACKEND


Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Find the detailed version of this roadmap along with resources and other roadmaps
http:// roadmap.sh
Backend
How does the internet work?
What is HTTP?
Browsers and how they work?
Internet
Basic Frontend Knowledge
HTML
DNS and how it works?
What is Domain Name?
What is hosting?
CSS
JavaScript
Terminal Usage
How OSs work in General
Process Management
Threads and Concurrency
Basic Terminal Commands
grep, awk, sed, lsof, curl, wget
tail, head, less, find, ssh, kill
OS and General Knowledge
Memory Management
Interprocess Communication
I/O Management
POSIX Basics
stdin, stdout, stderr, pipes
Basic Networking Concepts
Rust
Go
Java
C#
PHP
Learn a Language
Make sure to learn its quirks. Core details about its runtime e.g. concurrency, memory model etc.
JavaScript
Python
Ruby
Basic Usage of Git
Version Control Systems
What are they and why you should use one
Repo hosting services
Create account and Learn to use GitHub
GitHub
GitLab
Bitbucket

¿QUÉ VEREMOS HOY?

- Git: sistema de control de versiones.
- Gitlab: aplicación para administrar repositorios git y proyectos.
- Infraestructura de trabajo de la cátedra.


git

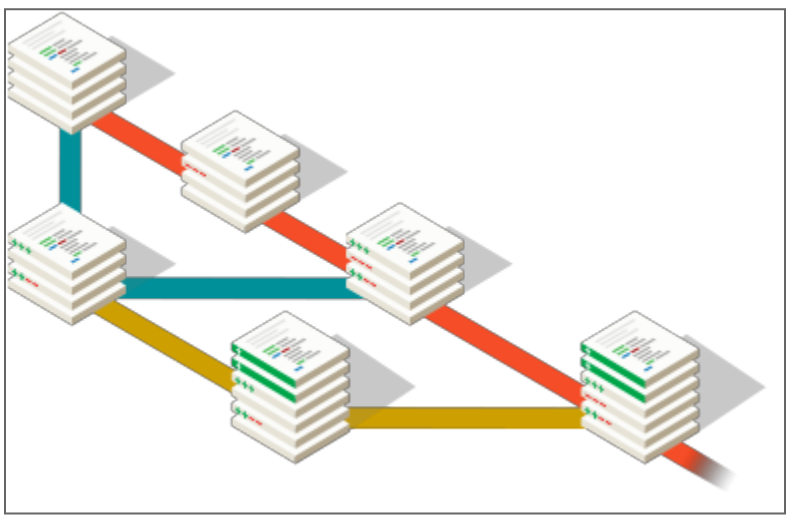
ENCUESTA INICIAL

¿Usan o usaron **git** anteriormente?

- **A** - Si, lo uso diariamente por trabajo o facultad.
- **B** - Si, ocasionalmente.
- **C** - No, pero sé que es.
- **D** - No sé que es y nunca lo usé.

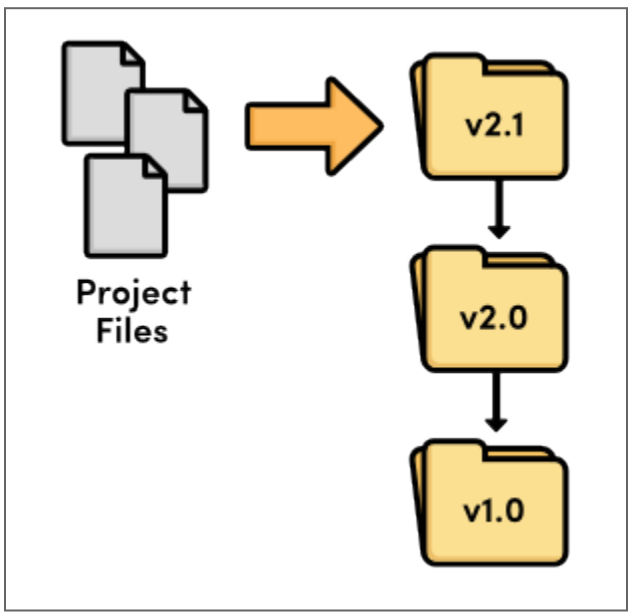
¿QUÉ ES UN SISTEMA DE CONTROL DE VERSIONES?

Es un sistema que registra los cambios realizados a nuestros archivos en el tiempo, de modo de poder volver a una versión anterior en cualquier momento.



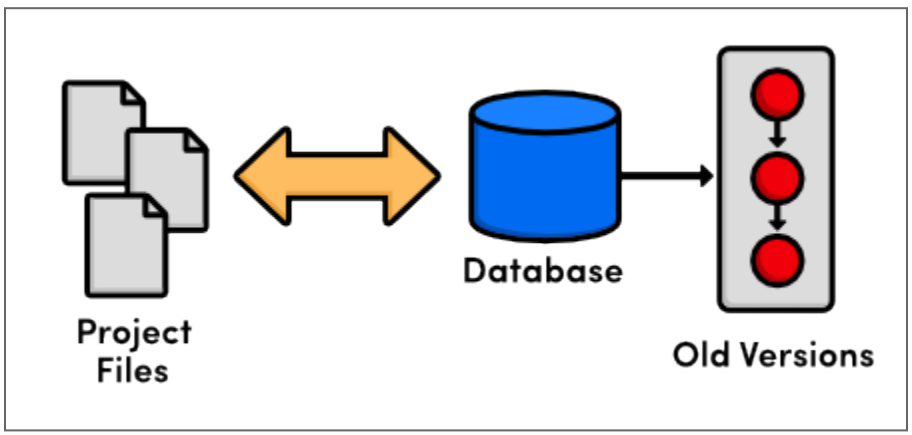
ANTES: SIN SISTEMA DE CONTROL DE VERSIONES

- La única forma era generando manualmente copias de los archivos y carpetas.



SISTEMA DE CONTROL DE VERSIONES LOCAL

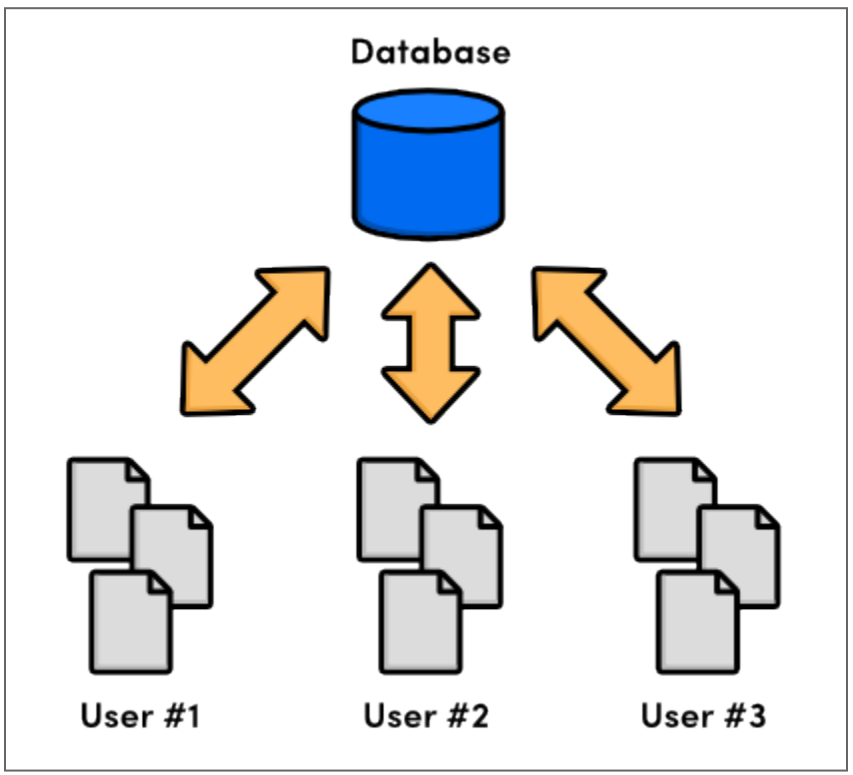
- Se comenzaron a desarrollar utilidades para manejar las revisiones localmente en una DB.



- Todas las operaciones son locales, compartir el trabajo con otro desarrollador era difícil.

SISTEMA DE CONTROL DE VERSIONES CENTRALIZADO

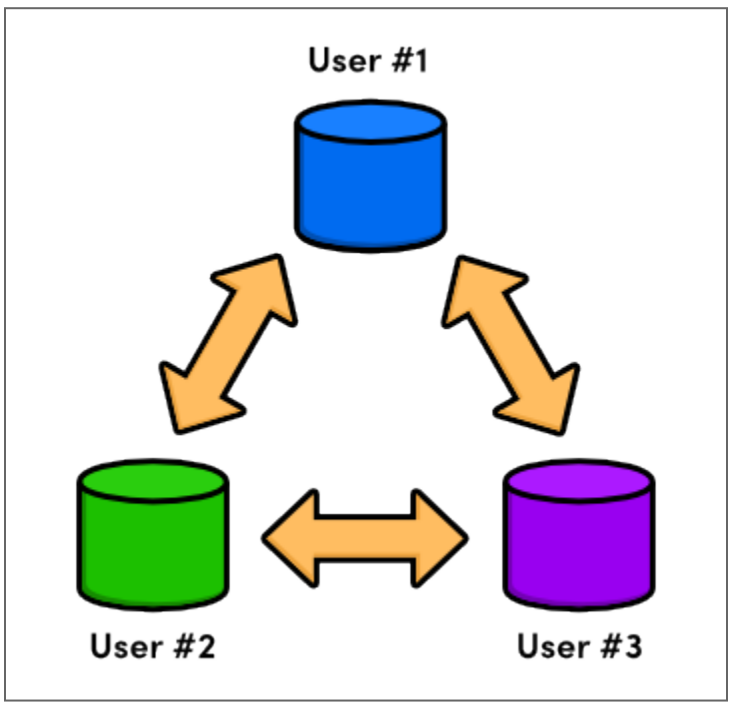
- En lugar de tener las revisiones en el disco del desarrollador, se tiene todo en un servidor centralizado.



- Los desarrolladores deben descargar y subir las nuevas versiones para compartirlas.

SISTEMA DE CONTROL DE VERSIONES DISTRIBUIDO

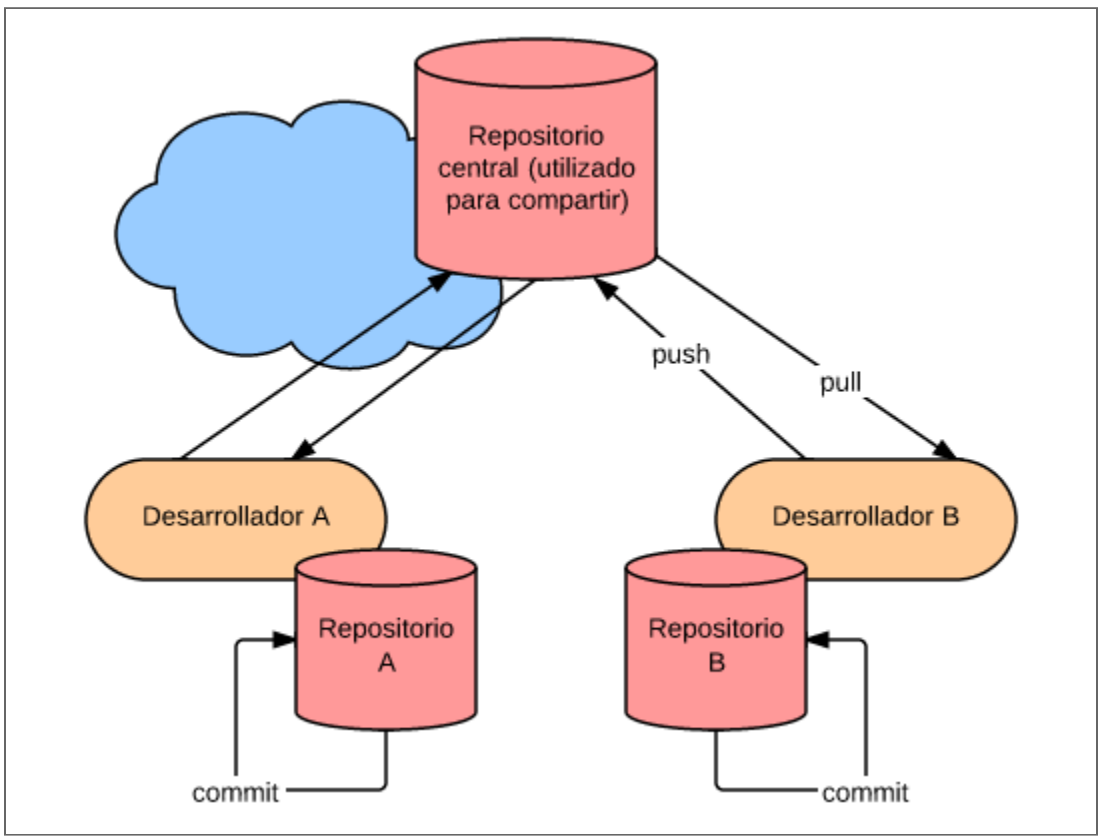
- Cada desarrollador tiene una copia de todo el repositorio, cada uno trabaja a su ritmo.



- Como la mayoría de las operaciones son ahora locales y no necesitan red, la velocidad de desarrollo se incrementó.

¿QUÉ ES GIT?

Git es un sistemas de control de versiones distribuido libre diseñado para manejar proyectos con velocidad y eficiencia.



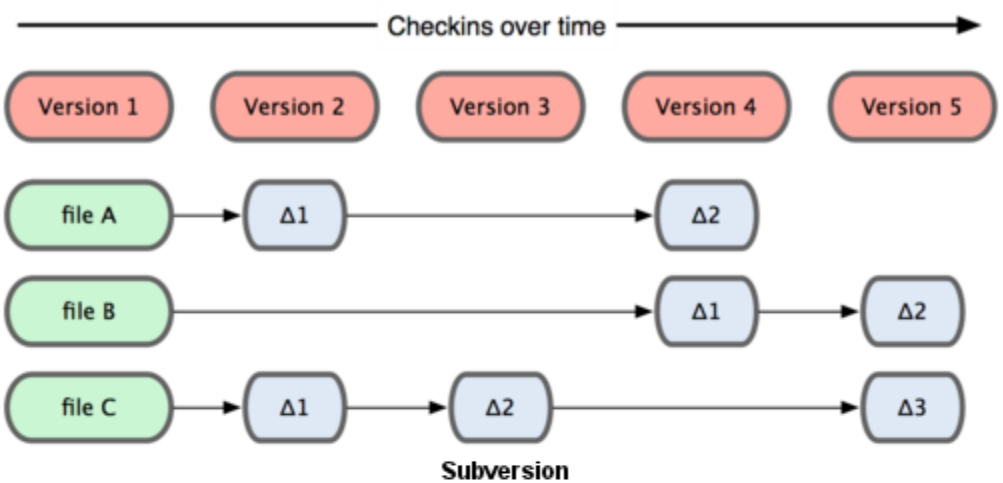
CARACTERÍSTICAS

- Snapshots, no diferencias
- Casi todas las operaciones son locales
- Tiene integridad

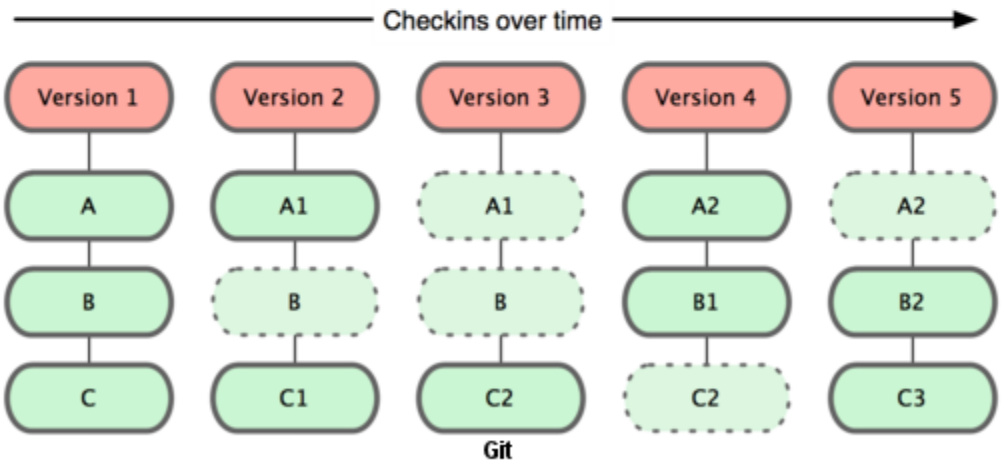
SNAPSHOTS, NO DIFERENCIAS

- La mayoría de los demás sistemas almacenan la información como una **lista de cambios** en los archivos.
- Git modela sus datos más como un **conjunto de instantáneas** (snapshots) de un mini sistema de archivos.
- Por eficiencia, si un archivo no cambió, no vuelve a guardarlo, sólo **referencia** al archivo ya almacenado.

SNAPSHOTS, NO DIFERENCIAS



Subversion



Git

CASI TODAS LAS OPERACIONES SON LOCALES

- La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para operar.
- Para navegar por la historia del proyecto, Git no necesita buscarla en el servidor.

TIENE INTEGRIDAD

- Todo en Git es verificado mediante una suma de comprobación antes de ser almacenado, y es identificado a partir de ese momento mediante dicho checksum.
- Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa.
- Ejemplo (**SHA-1 de 40 caracteres**):

```
ea36b870f9a0e1e6439758b6e681bd329a04db3d
```

INSTALACIÓN DE GIT

Linux (Debian/Ubuntu)

```
# apt-get install git
```

Linux (Arch)

```
# pacman -S git
```

Linux (Fedora)

```
# yum install git
```

- Mac: **https://git-scm.com/download/mac**
- Windows: **https://git-scm.com/download/win**
- Integrado con VScode: **https://code.visualstudio.com/Docs/editor/versioncontrol**

- Integrado con Eclipse: **http://www.eclipse.org/egit/**

OBTENIENDO UN REPOSITORIO GIT

Inicializar un repositorio en un directorio existente

```
$ git init
```

Clonando un repositorio existente

```
$ git clone
gitlab@git.proyecto.linti.unlp.edu.ar/grupo.git
```

DIRECTORIO .GIT/

- Cada repositorio Git es almacenado en la carpeta **.git** del directorio en el cual el repositorio ha sido creado.
- Este directorio contiene la historia completa del repositorio. El archivo **.git/config** contiene la configuración local del repositorio.

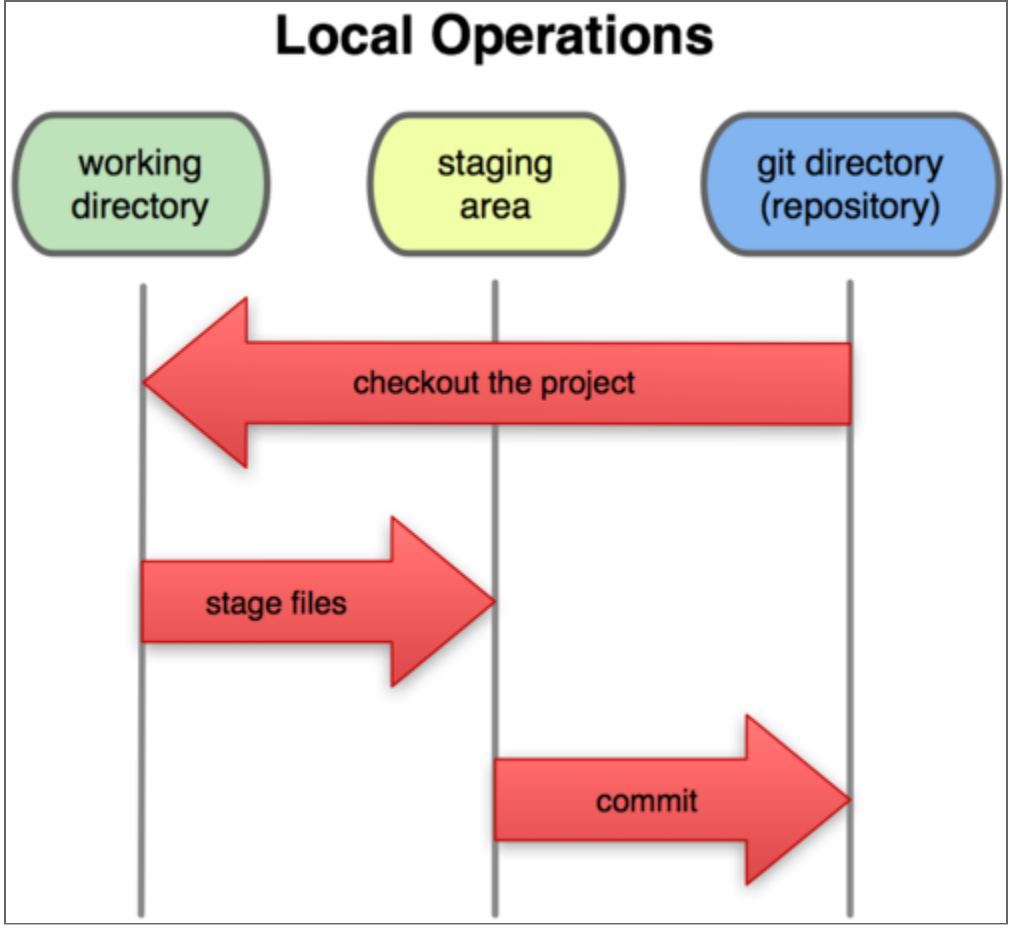
CONFIGURACIÓN DEL USUARIO

Configurá tu usuario y mail para Git mediante los siguientes comandos:

```
# Configura el usuario que será usado por git
# Obviamente deberías usar tu nombre
git config --global user.name "John Doe"
# Lo mismo para el correo electrónico
git config --global user.email "jdoe@example.com"
```

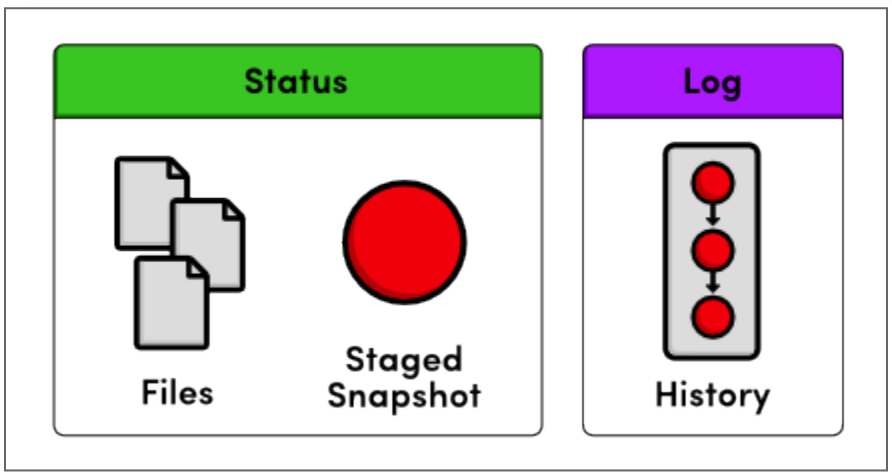
OPERACIONES LOCALES

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged).


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit

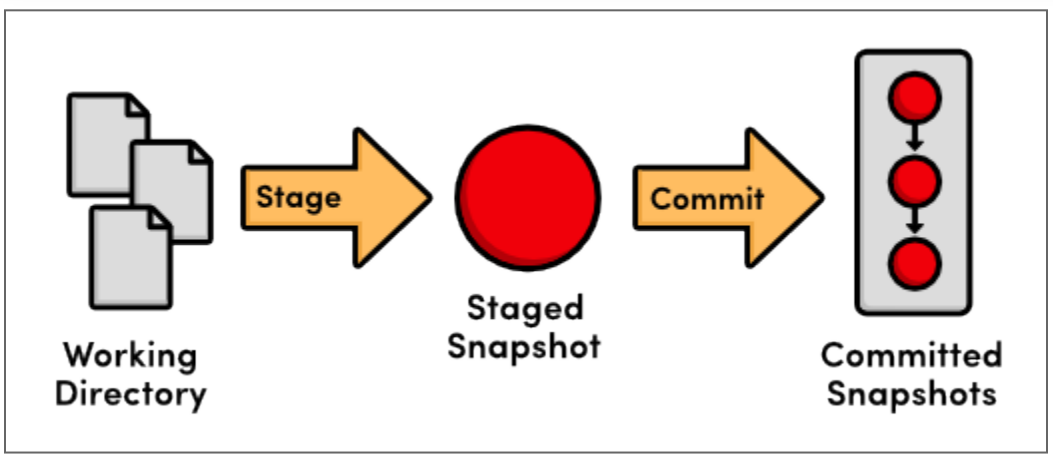
COMPROBANDO EL ESTADO DEL DIRECTORIO DE TRABAJO

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

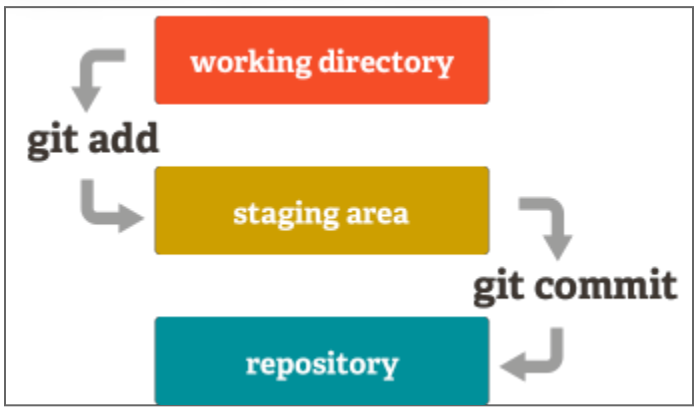


CREANDO Y MODIFICANDO CONTENIDO

```
# Creamos contenido
# Agregamos todo (archivos y directorios) al repositorio
$ git add .
# Hacemos un commit al repositorio
$ git commit -m "Initial commit"
# Muestra el log (un historial)
$ git log
```



CICLO DE VIDA DE COMMITS



VIENDO LAS MODIFICACIONES

El comando **git diff** permite al usuario, entre otras cosas, ver los cambios hechos desde el último commit.

```
# Mirá los cambios con el comando diff
git diff
# Comitea con -a sube los cambios de los archivos
# pero no agrega automaticamente nuevos archivos
git commit -a -m "Hay nuevos cambios"
```

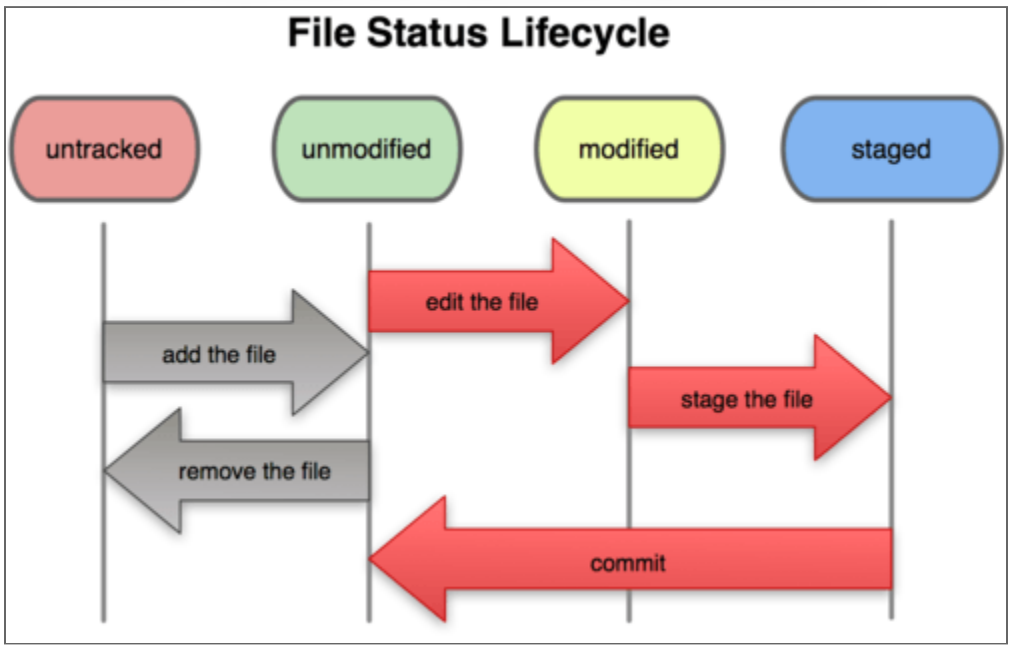
Comparar commits

```
git diff df2db72c 18e19e7a
```

Comparar ramas

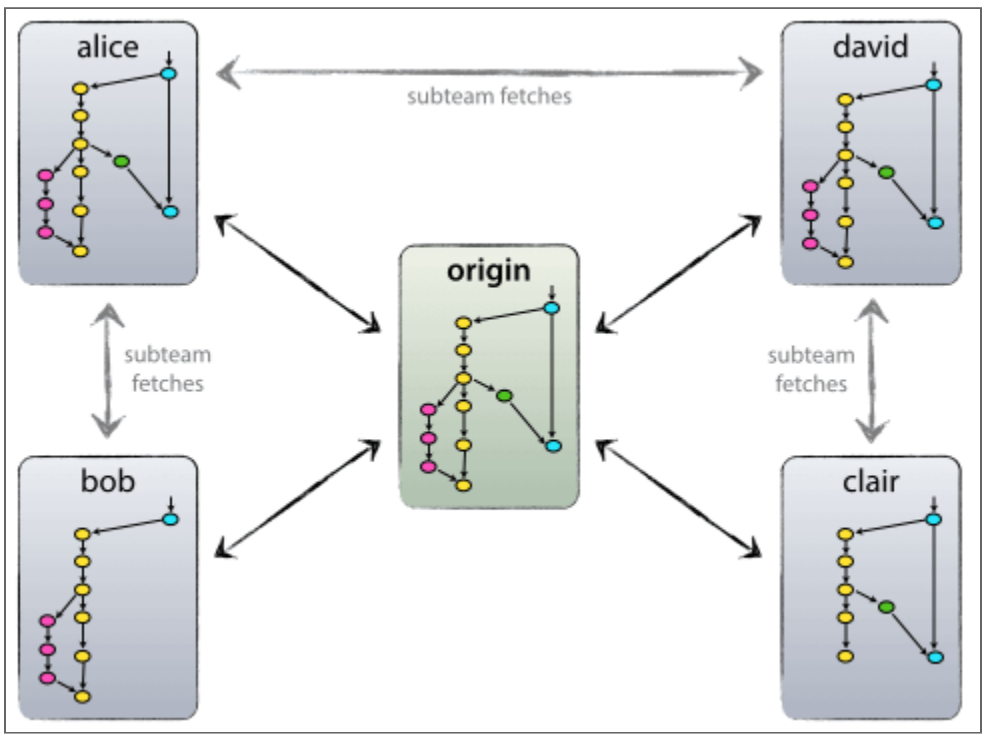
```
git diff develop..master
```

CICLO DE VIDA DE UN ARCHIVO LOCALMENTE



REPOSITORIOS REMOTOS

- Son repositorios externos (ejemplo: de coworker, gitlab, github, etc).
- Puede haber n remotos para un repositorio git.



TRABAJANDO CON REPOSITORIOS REMOTOS

Viendo los remotos actuales:

```
$ git remote
origin

$ git remote -v
origin  git@gitlab.com:proyecto/www.git (fetch)
origin  git@gitlab.com:proyecto/www.git (push)
```

Agregando un nuevo repositorio remoto:

```
$ git remote add shortname url
```

TRABAJANDO CON REPOSITORIOS REMOTOS

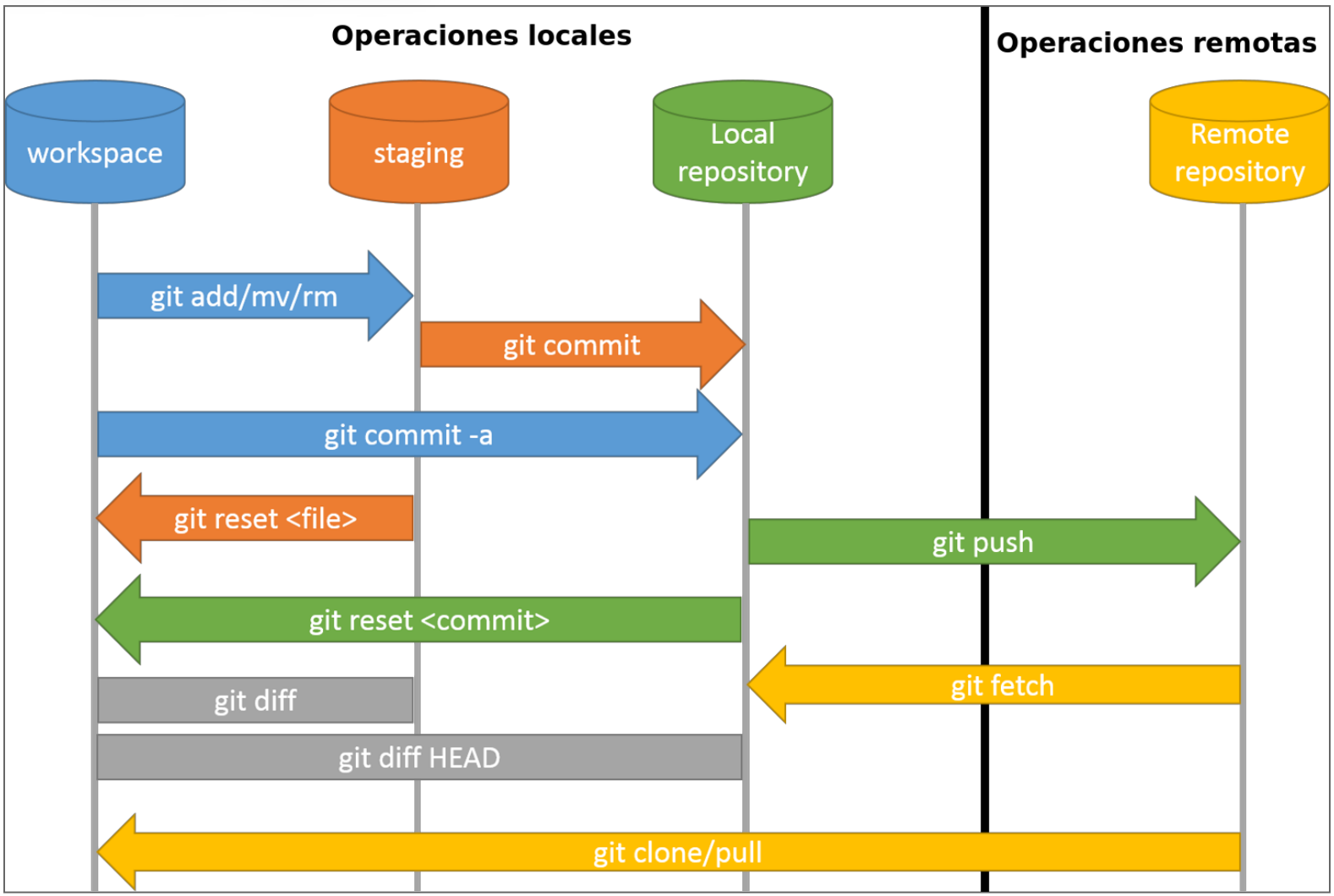
Obteniendo cambios del remoto:

```
$ git fetch remote
$ git pull remote
```

Subiendo cambios al remoto

```
$ git push origin master
```

EL WORKFLOW COMPLETO



REFERENCIAS DE GIT:

- Git: **http://git-scm.com/**
- Libro: Pro Git: **http://git-scm.com/book**
- Libro: Ry's Git Tutorial: **https://www.smashwords.com/books/view/498426**
- Git Cheatsheet (Github): **https://education.github.com/git-cheat-sheet-education.pdf**
- Git Cheatsheet en Español: **https://github.com/arslanbilal/git-cheat-sheet/blob/master/other-sheets/git-cheat-sheet-es.md**
- Git Cheatsheet interactivo: **http://www.ndpsoftware.com/git-cheatsheet.html**

¿ALGUNA CONSULTA HASTA ACÁ DE GIT?

Continúa en video...


GitLab

¿QUÉ ES GITLAB?

- GitLab es una aplicación opensource que nos permite administrar repositorios en git mediante una interfaz web.
- Es un clon de **http://github.com** y es una herramienta muy potente para el desarrollo.

ANTES QUE NADA ...

- GitLab utiliza **claves SSH** para permitir trabajar con los respositorios.
- Las claves SSH son utlizadas para establecer una conexión segura entre los repositorios y GitLab.
- Con lo cual lo primero que necesitamos hacer es subir nuestra clave **pública** al proyecto.
- Si no realizamos esto el usuario no podrá subir los cambios realizado en su repositorio local al proyecto de GitLab!!

GENERANDO NUEVAS CLAVES SSH

```
ssh-keygen -t rsa -C "your_email@example.com"
```

AGREGANDO LA CLAVE SSH

- Ir a a la sección de claves SSh de su perfil del usuario **https://gitlab.catedras.linti.unlp.edu.ar/-/profile/keys**
- Agregar la clave pública

```
.ssh/id_rsa.pub
```

- Se va a utilizar para identificar y autenticar cada interacción con el servidor

CONFIGURANDO EL USUARIO DEL REPOSITORIO

```
git config --global user.name "John Doe"
git config --global user.email "johndoe@mail.com"
```

INICIALIZANDO EL REPOSITORIO GIT DE NUESTRO PROYECTO GITLAB

- En GitLab inicialmente tenemos un proyecto que no tiene un repositorio local asignado.
- Tenemos dos opciones:
  - Crear un repositorio vacío y enlazarlo al repositorio local de nuestro proyecto en GitLab.
  - Utilizar un repositorio ya creado y sólo debemos asignarlo al proyecto de GitLab.

CREAR UN REPOSITORIO VACÍO

Es la opción más común.

```
mkdir grupo_1
cd grupo_1
git init
touch README
git add README
git commit -m 'first commit'
git remote add origin
gitlab@proyecto.linti.unlp.edu.ar:/grupo_1.git
git push -u origin master
```

UTILIZANDO GITLAB

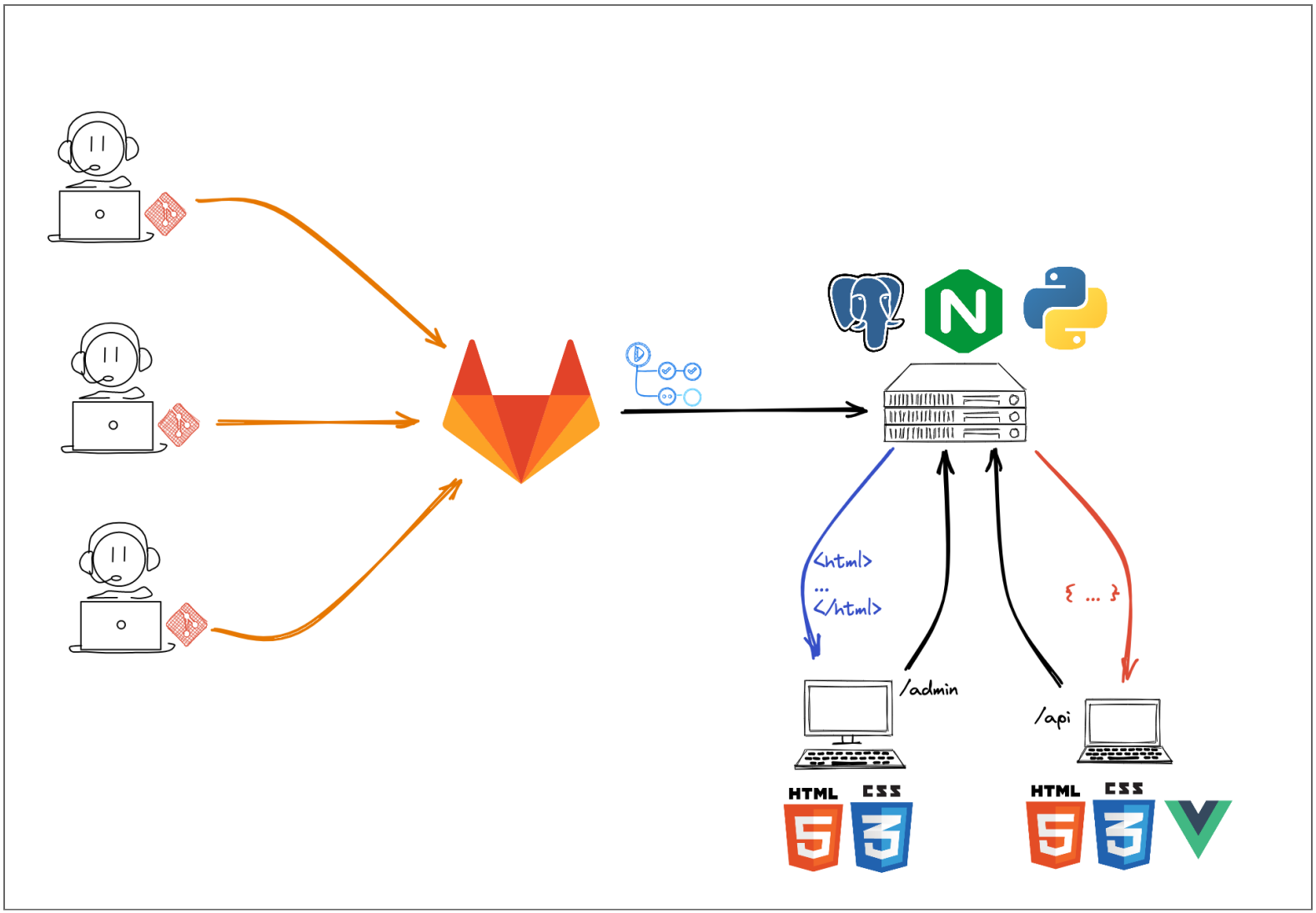
Teniendo nuestro respositorio git creado, podemos empezar a utilizarlo y en GitLab tener:

- El último estado de nuestros archivos.
- Seguimiento de los **commits** realizados y las diferencias aplicadas.
- Una red con el crecimiento de las versiones y bifurcaciones que va tomando nuestro repositorio.
- Gráficos con estadísticas de uso.
- Creación y seguimiento de tareas (o **issues**) relativas al proyecto.
- Una **wiki** con información propia de cada proyecto.

DEMO

- https://gitlab.catedras.linti.unlp.edu.ar/

INFRAESTRUCTURA DE TRABAJO DE LA CÁTEDRA


<html>
...
</html>
{ ... }
/admin
/api
HTML
CSS
HTML
CSS

CONSULTAS FINALES

FIN