Introducción a
GIT

Conceptos Clave:

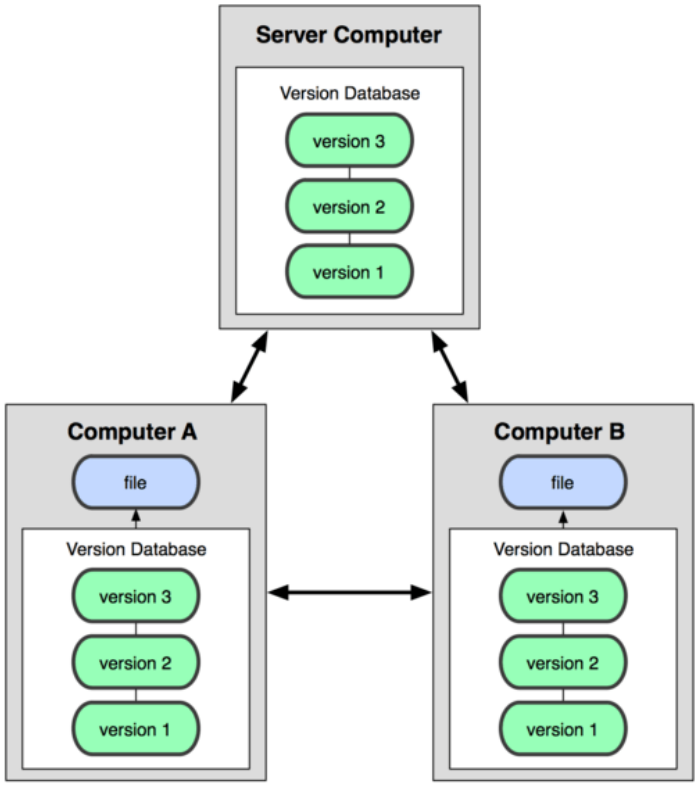
- ¿Que es el Control de Versiones?
- Sistemas Centralizados vs Sistemas Distribuidos
- Qué es GIT

¿Qué es un Sistema Control de Versiones?

- Los sistemas de control de versiones son una categoría de herramientas de software que ayudan a un equipo de software a gestionar los cambios en el código fuente a lo largo del tiempo. El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

Sistemas Centralizados vs Sistemas Distribuidos

- **Los Sistemas de Control de Versiones Centralizados (VCS)**, como por ejemplo Subversión, que es una herramienta en la que se ha confiado para albergar el histórico de revisión de versiones, es un punto centralizado, lo cual puede llegar a suponer una merma de trabajo si perdemos la conectividad de la red.
- **Los Sistemas de Control de Versiones Distribuidos (DVCS)** salvan este problema. Algunos ejemplos de sistemas distribuidos, aparte de Git, son **Mercurial, Bazaar o Darcs**. En este tipo de herramientas, los clientes replican completamente el repositorio.



En **Sistemas de Control de Versiones Centralizados**, como Subversion, partimos de una versión, por ejemplo de la versión 1 y tenemos tres ficheros, A, B y C.
De la versión 1 a la versión 2, las diferencias, como si fueran esa especie de incrementos que llamamos deltas, son almacenados por el sistema. De esta manera, por buscar algún símil, es como si Subversion trabajase con backups incrementales.

Después tenemos el funcionamiento de Sistemas de Control de Versiones de Git, en el que cada vez que hay cambios de ficheros éste es almacenado otra vez, y si no hay cambios es como si tuviésemos una especie de apuntador al fichero que no ha tenido cambios, en una revisión o en un hito del tiempo anterior.

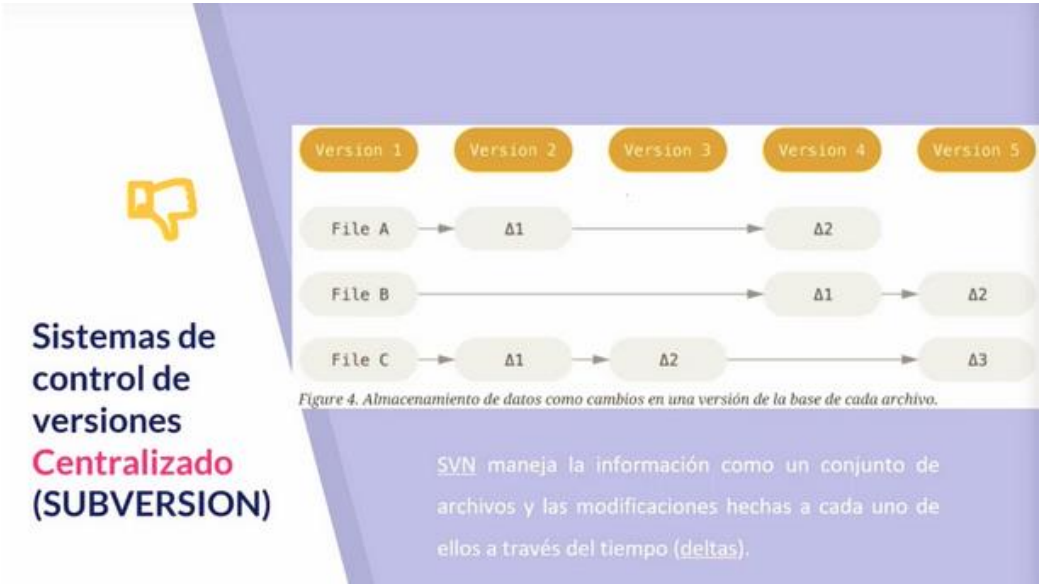


Figure 4. Almacenamiento de datos como cambios en una versión de la base de cada archivo.

Qué es GIT

```
git config --global user.name "John Doe“

git config --global user.email
```

Los 3 estados de GIT

confirmado (*committed*), **modificado** (*modified*) y **preparado** (*staged*)

1. Hago cambios | 2. git add | 3. git commit | 4. Vuelvo a empezar

Comandos básicos

Inicializando un repositorio

`git init` **(initialize** a new **git** repository)

`git clone` (create a **copy** of an existing repository)

Comandos básicos

Del Working Directory al Staging Area

```
git add <filename>
git add <filename1> <filenameN>
git add .
```

Comandos básicos

Del Staging Area al Repository

```
git commit –m “Commit Message”
```

Standard Conventions for Commit Messages:

- Must be in quotation marks
- Written in the present tense
- Should be brief (50 characters or less) when using `-m`

Comandos básicos

Viendo el estado de los archivos

```
git status
```

Comandos básicos

Revisando la historia de nuestro proyecto

```
git log
```

Comandos básicos

Revisando las diferencias entre el working directory y el staging área (cambios desde el último commit)

```
git diff <filename>
```

Comandos básicos

Revisando las diferencias de un archivo entre dos Branch diferentes

```
git diff branch1 branch2 <filename>
```

Comandos básicos

Revisando las diferencias de un archivo entre dos commits

```
git diff commit1 commit2 <filename>
```

Comandos básicos

RETROCESO

Descartando cambios del working directory

```
git checkout -- <filename>
git checkout -- <filename1>
<filenameN>
git checkout -- .
```

Comandos básicos

RETROCESO

Deshaciendo commits

`git reset --hard <hash>` (Deshaciendo el commit perdiendo las modificaciones)

`git reset <hash>` (Deshaciendo el commit manteniendo las modificaciones)

`git log` (Para obtener el hash del commit)

Comandos básicos

¿En qué branch/rama estoy?

```
git branch
```

Comandos básicos

Creando una nueva rama

```
git checkout -b "new_branch"
```

*branch names can't contain whitespaces
new_branch
new-branch

Comandos básicos

Moviendo a otra rama

```
git checkout “branch_name”
```

Comandos básicos

Guardado rápido provisional

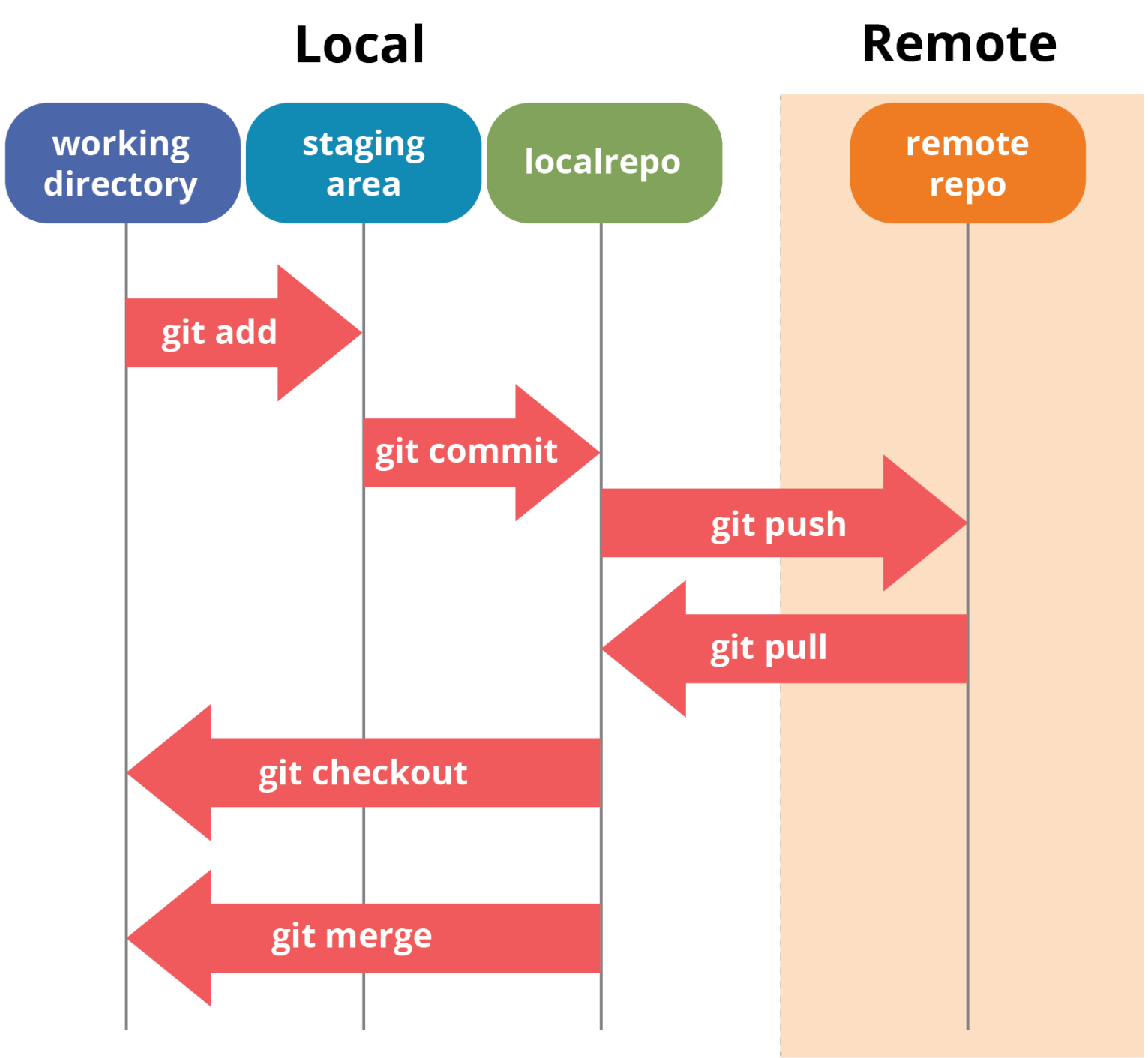
```
git stash
git stash list
git stash pop
```

.gitignore

Ignorando archivos no deseados

¿Qué es GitHub?

- **Plataforma de desarrollo colaborativo**
- **Aloja proyectos utilizando el sistema de control de versiones Git**
- **Público y Privado (de pago)**
- **El 4 de junio de 2018, Microsoft compró GitHub por la cantidad de 7.500 millones de dólares.**


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout
git merge

Comandos básicos

Trayendo cambios desde el repositorio remoto

```
git pull
```

Comandos básicos

Enviando cambios al repositorio remoto

```
git push
```

```
git push –u origin <branch>
```

(Añadir un branch local a remote)

Comandos básicos

Eliminando un Branch de forma local y remote

```
git push origin –delete <branch_name>
```

(remote)

```
git branch –d <branch_name>
```

(local)

Comandos básicos

Combinando archivos de diferentes ramas

```
git merge
```

QUESTION 9

Which command should you use to initialize a new Git repository?

- `git bash`
- `git install`
- `git init`
- `git start`

QUESTION 10

Which file can you configure to ensure that certain file types are never committed to the local Git repository?

- *ignore.git*
- *.gitignore*
- *gitignore.txt*
- *git.ignore*

4. We've just created a new file called `index.html`. Which of the following will stage this one file so we can commit it?

```
git add index.html
```

```
git add new
```

```
git commit index.html
```

What command do you run to view the commit history of your repository?

- ○ git history
- ○ git log
- ○ git commit -h
- ○ git past

Hoja de Referencia

https://services.github.com/on-demand/downloads/es_ES/github-git-cheat-sheet.pdf

¡A PRACTICAR!
¡A PRACTICAR!
¡A PRACTICAR!

https://github.com/eduatro/simple-on-line-status