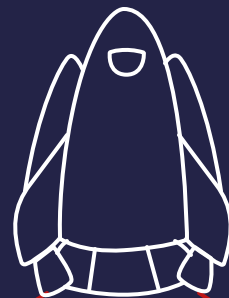


# Programa académico CAMPUS



Ciclo 2:  
GIT  
INTRODUCCIÓN



# GIT - INTRODUCCIÓN



# GIT - INTRODUCCIÓN



## Sistemas de control de versiones (VCS)

Los **VCS** (*del inglés Version Control System*) ayuda a **registrar los cambios** en un archivo o grupo de archivos. Por lo tanto, podemos **recuperar una versión** específica de un archivo en momento determinado.



# GIT - INTRODUCCIÓN



## Sistemas de control de versiones (VCS)

Generalmente se usan para **controlar los cambios** del código fuente de una aplicación. No obstante, puedes usarlos para administrar cualquier tipo de documento.



# GIT - INTRODUCCIÓN



## Sistemas de control de versiones (VCS)



Estos trabajan al nivel del sistema de archivo. En consecuencia, *monitorean cambios*, adiciones y eliminación de archivos y/o carpetas.



# GIT - INTRODUCCIÓN



## Ventajas de usar VCS



-  Guarda el registro histórico de las modificaciones realizadas a cada archivo y/o carpeta.
-  Añade trazabilidad al desarrollo de software. Por lo tanto, permite identificar cambios entre una versión y otra.



# GIT - INTRODUCCIÓN



## Ventajas de usar VCS



-  Colaboración de varios desarrolladores en el mismo proyecto.
-  Ayudan a gestionar y unir los diferentes archivos del proyecto.



# GIT - INTRODUCCIÓN



## Ventajas de usar VCS

-  Evita que pierdas información que por error se haya modifica o borrado accidentalmente.
-  Es posible revertir o deshacer los cambios realizados en el archivo.





# GIT - INTRODUCCIÓN



## ARQUITECTURA DE LOS VCS

Los VCS han evolucionado como los programadores lo han requerido. Desde las copias locales, hasta complejos sistemas que permiten la colaboración de varios programadores en un mismo proyecto.



# GIT - INTRODUCCIÓN



## ARQUITECTURA VCS - LOCALES

Consiste en *crear copias locales* de tus archivos en otra carpeta de tu computadora.

Este sistema se caracteriza por ser fácil de implementar. Es decir, copias, pegas y renombros directorios.



# GIT - INTRODUCCIÓN



## ARQUITECTURA VCS - LOCALES

Su *principal desventaja* radica en la poca o nula trazabilidad que se puede dar a cada archivo.

Si accidentalmente borraras una carpeta que contenía X versión de tu código, no será posible recuperar esa información.



# GIT - INTRODUCCIÓN



## ARQUITECTURA VCS - LOCALES

- La combinación de archivos puede ser casi imposible. Aunque puedes hacer uso de ciertas herramientas (enlace herramientas) para combinar archivos, perderás el registro de tus cambios.



# GIT - INTRODUCCIÓN



## ARQUITECTURA VCS - LOCALES

- Al ser un sistema local, es imposible que alguien más colabore en tus proyectos. Al menos que quieras prestarle tu computadora a alguien más.

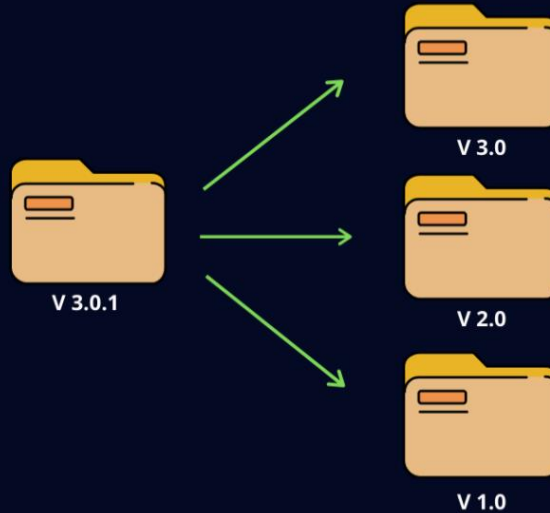


# GIT - INTRODUCCIÓN



## Locales

Copias locales del código fuente.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - CENTRALIZADOS

- Ahora, pasamos a las copias en un servidor dedicado.

Es decir, un sistema centralizado es un sistema cliente – servidor.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - CENTRALIZADOS

- Lo sistemas centralizados se caracterizan por almacenar una copia del código fuente en una base de datos central.

Por consiguiente, cualquier usuario que tenga acceso a la base de datos, también puede

hacer una copia y modificar los archivos.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - CENTRALIZADOS

- Cuando el usuario termina de modificar el archivo, solicita al control de versiones que aplique los cambios en la versión del servidor.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - CENTRALIZADOS

- Pero no todo es hermoso. Si dos programadores trabajan en el mismo archivo. Solo el primero que envíe sus cambios podrá hacerlo de forma limpia.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - CENTRALIZADOS

- El segundo tendrá que fusionar los archivos de forma manual. Solo hasta que están integrados los cambios el resto de los programadores podrán continuar trabajando.

Por otra parte, si el servidor central falla, nadie podrá colaborar en el proyecto.

# GIT - INTRODUCCIÓN



## Centralizados

Una **copia central** en servidor dedicado. (Cliente - Servidor)



# GIT - INTRODUCCIÓN



- **ARQUITECTURA VCS - DISTRIBUIDOS**
- Tiene una copia central, pero cada programador puede replica una copia local del código fuente.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - DISTRIBUIDOS

- Por lo tanto, cada programador puede trabajar de forma aislada. Asimismo, este sistema cuenta con herramientas que facilitan la resolución de conflictos.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - DISTRIBUIDOS

- Al ser un sistema distribuido, no requieres estar conectado al servidor central para poder trabajar.



# GIT - INTRODUCCIÓN



## • ARQUITECTURA VCS - DISTRIBUIDOS

- Aunque permiten la colaboración de múltiples programadores. Solo el administrador de repositorio puede aceptar o rechazar cambios.
- Por supuesto, al permitirse múltiples copias de código fuente, se puede perder el control de quién tiene copias de este.



# GIT - INTRODUCCIÓN



## Distribuidos.

Cada programador tiene una copia local del código.



# GIT - INTRODUCCIÓN



## DIFERENTES VCS

15 BEST Version Control Software -  
<https://bit.ly/15-VCS>



TOP VERSION CONTROL SYSTEMS

# GIT - INTRODUCCIÓN



## Bazaar

Arquitectura	Distribuida
Fecha de lanzamiento	26 de marzo de 2005
Programado en	Python
Desarrollado por	Canonical Proyecto GNU
Autor	Martin Pool



# GIT - INTRODUCCIÓN



## Apache Subversion

Arquitectura	Cliente – Servidor
Fecha de lanzamiento	20 de octubre de 2000
Programado en	C
Desarrollado por	Apache Software Foundation
Autor	CollabNet, Inc.



# GIT - INTRODUCCIÓN



## Mercurial

Arquitectura	Distribuida
Fecha de lanzamiento	19 de abril de 2005
Programado en	Mayormente en Python con pequeñas partes portables en C.
Autor	Matt Mackall



# GIT - INTRODUCCIÓN



## GIT

Arquitectura	Distribuida
Fecha de lanzamiento	7 de abril de 2005
Programado en	C, Bourne Shell, Perl
Desarrollado por	Linus Torvalds, Junio Hamano, Software Freedom Conservancy
Autor	Linus Torvalds



# GIT - INTRODUCCIÓN

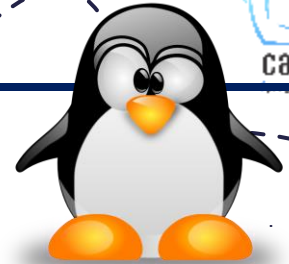


## GIT - HISTORIA

- Git fue impulsado por Linus Torvalds y el equipo de desarrollo del Kernel de Linux. Ellos estaban usando otro sistema de control de versiones de código abierto, que ya por aquel entonces era distribuido.



# GIT - INTRODUCCIÓN



## GIT - HISTORIA

- Todo iba bien hasta que los gestores de aquel sistema de control de versiones lo convirtieron en un software propietario. Lógicamente, no era compatible estar construyendo un sistema de código abierto, tan representativo como el núcleo de Linux, y estar pagando por usar un sistema de control de versiones propietario.



# GIT - INTRODUCCIÓN



## GIT - HISTORIA



- Por ello, el mismo equipo de desarrollo del Kernel de Linux se tomó la tarea de construir desde cero un sistema de versionado de software, también distribuido, que aportase lo mejor de los sistemas existentes hasta el momento.



# GIT - INTRODUCCIÓN



## GIT - AREAS

- Entender el funcionamiento de las tres zonas es lo más importante de GIT, si consigues entender adecuadamente este apartado, el resto de contenido que veremos acerca de git te parecerá muy sencillo.

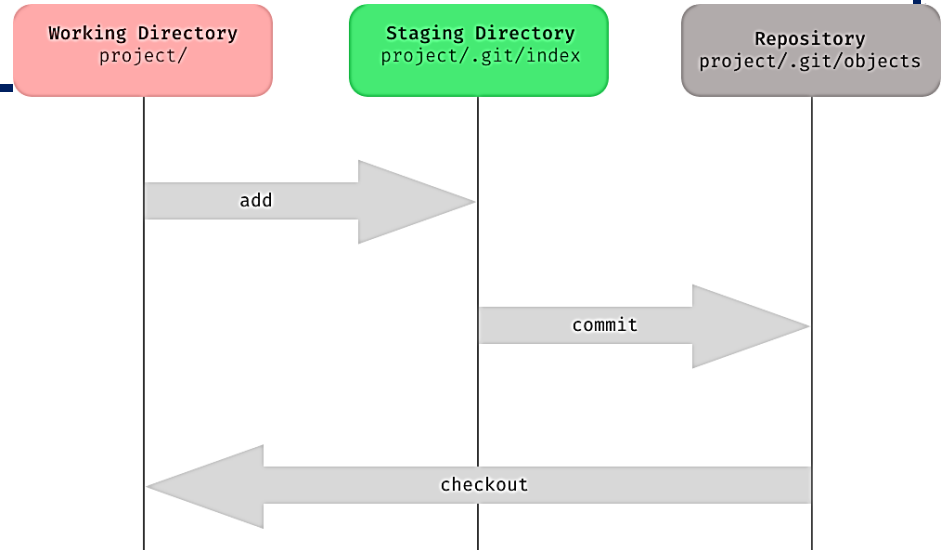


# GIT - INTRODUCCIÓN



## GIT - AREAS

En git existen tres zonas principales que se deben destacar.

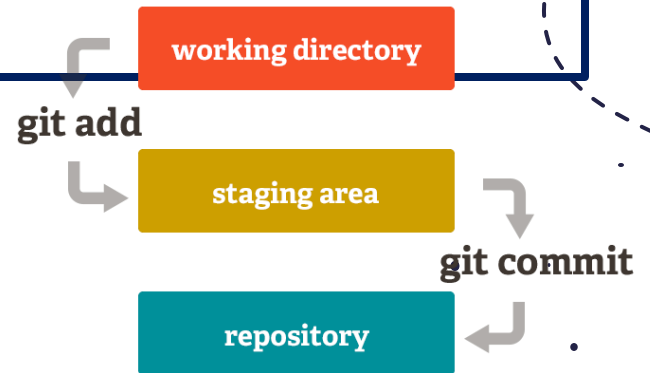


# GIT - INTRODUCCIÓN



## GIT - AREAS

*El Directorio de trabajo* es considerado el directorio donde inicializamos git. En esta zona los ficheros están en el estado Untracked (Que no están en seguimiento)



# GIT - INTRODUCCIÓN



## GIT - AREAS

La ***zona de preparación (staging)*** es donde colocamos los ficheros que deseamos tener en seguimiento, cuando los ficheros se encuentran en esta zona, git nos indicará si el fichero es nuevo para él, si un fichero ha sido modificado o si ha sido borrado por completo para poder deshacer esa operación local incluso antes de confirmar los cambios.

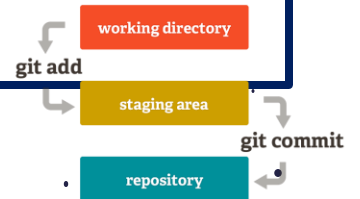


# GIT - INTRODUCCIÓN

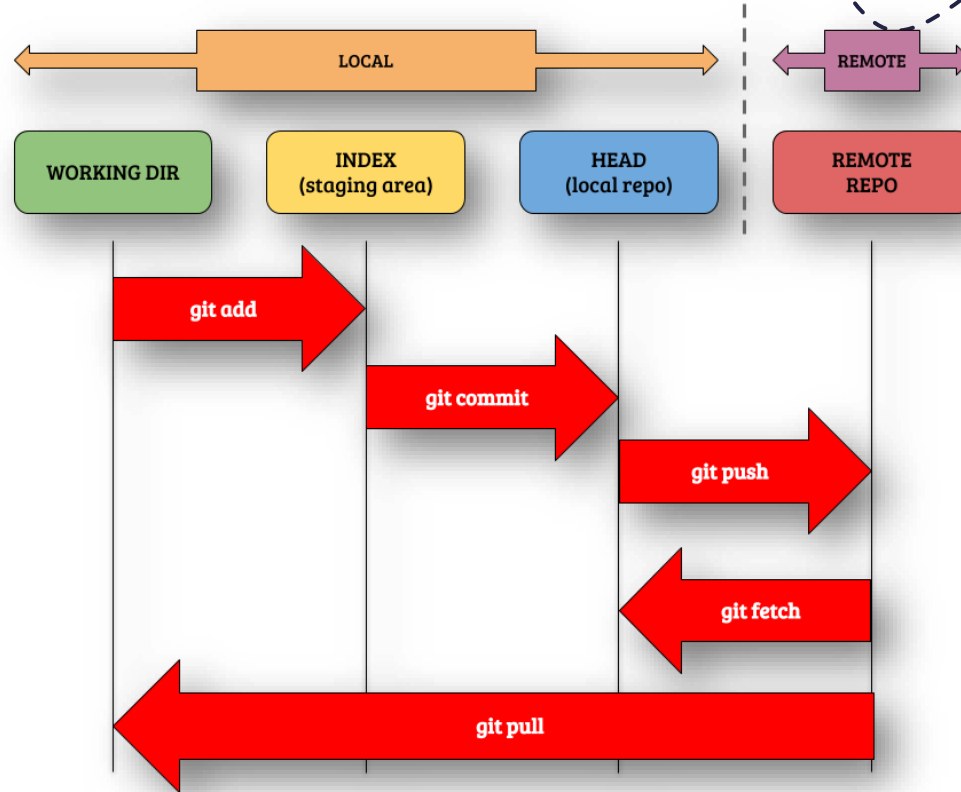


## GIT - AREAS

el **repositorio git** o **zona de commits** son los cambios confirmados después de las verificaciones a través de la zona de preparación. Estos cambios se aplicaran directamente sobre el directorio de trabajo y el ciclo empezará de nuevo.



# GIT - INTRODUCCIÓN



# GIT - INTRODUCCIÓN



## GIT - INSTALACIÓN - Windows

Git - Downloading Package (git-scm.com)

### Download for Windows

[Click here to download](#) the latest (2.39.2) 64-bit version of **Git for Windows**. This is the most recent [maintained build](#). It was released **22 days ago**, on 2023-02-14.

#### Other Git for Windows downloads

Standalone Installer

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)





# GIT - INTRODUCCIÓN



## GIT – INSTALACIÓN – Linux Ubuntu

Git (git-scm.com)

### Download for Linux and Unix

It is easiest to install Git on Linux using the preferred package manager of your Linux distribution. If you prefer to build from source, you can find tarballs [on kernel.org](https://kernel.org). The latest version is **2.39.2**.

#### Debian/Ubuntu

For the latest stable version for your release of Debian/Ubuntu

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

For Ubuntu, this PPA provides the latest stable upstream Git version

```
# add-apt-repository ppa:git-core/ppa # apt update; apt install git
```



# GIT - INTRODUCCIÓN



## GIT – CONFIGURAR

**git config --list**

Este comando comprueba la configuración actual.

```
ubuntu@literary-manta:~/aprendiendo-git$ git config --list
user.name=tito
user.email=carloshrueda@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```



# GIT - INTRODUCCIÓN



## GIT – CONFIGURAR - Usuario

`git config --global user.name "[user]"`

Este comando configura el usuario de git

```
$ git config --global user.name carlos|
```



# GIT - INTRODUCCIÓN



- **GIT – Creación directorio de trabajo**

**mkdir** *"[directorio]"*

- Comando de Linux para crear un directorio

```
t$ mkdir aprendiendo-git|
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git init**

Se inicia un repositorio en Git (inicia el rastreo de git).

Esto creará el staging o área de ensayo y un repositorio local.



# GIT - INTRODUCCIÓN



ubuntu@literary-manta: ~/api

```
ubuntu@literary-manta:~/aprendiendo-git$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ubuntu/aprendiendo-git/.git/
ubuntu@literary-manta:~/aprendiendo-git$
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**tree .git**

Comando Linux para ver el árbol de directorio oculto de .git. Este se crea al usar el git init.



# GIT - INTRODUCCIÓN



```
ubuntu@literary-manta:~/aprendiendo-git$ tree .git
.git
|-- HEAD
|-- branches
|-- config
|-- description
|-- hooks
|   |-- applypatch-msg.sample
|   |-- commit-msg.sample
|   |-- fsmonitor-watchman.sample
|   |-- post-update.sample
|   |-- pre-applypatch.sample
|   |-- pre-commit.sample
|   |-- pre-merge-commit.sample
|   |-- pre-push.sample
|   |-- pre-rebase.sample
|   |-- pre-receive.sample
|   |-- prepare-commit-msg.sample
|   |-- push-to-checkout.sample
|   |-- update.sample
|-- info
|   |-- exclude
|-- objects
|   |-- info
|   |-- pack
|-- refs
|   |-- heads
|   |-- tags
```





# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

### git status

Con este comando lo que hacemos es, ver el estatus de nuestro repositorio(es decir los archivos modificados en el `working directory`).

```
ubuntu@literary-manta: ~/api  ×  +  v
ubuntu@literary-manta:~/aprendiendo-git$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
ubuntu@literary-manta:~/aprendiendo-git$ |
```



# GIT - INTRODUCCIÓN



Delta  
Changes:  
- A new file  
- File changes

I  
WORKING  
DIRECTORY

II  
STAGE AREA

III  
LOCAL  
REPOSITORY



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

`vim index.html`

Editor de texto de línea de comando de sistemas Unix.

A terminal window with a dark background and light blue text. The window title bar shows 'ubuntu@literary-manta: ~/api' and a search icon. The text in the terminal is HTML code: 

```
<html>
  <head>
</head>

  <body>
</body>
</html>
```

 The cursor is at the end of the last line.

# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**ls -l**

Comando Linux para listar el contenido de un directorio.

```
ubuntu@literary-manta: ~/api  X  +  v
ubuntu@literary-manta:~/aprendiendo-git$ ls -l
total 4
-rw-rw-r-- 1 ubuntu ubuntu 50 Mar  8 12:20 index.html
ubuntu@literary-manta:~/aprendiendo-git$
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

`git status`

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
index.html
```

```
nothing added to commit but untracked files present (use "git add" to track)
```



# GIT - INTRODUCCIÓN



Create a new file inside the git repository



# GIT - INTRODUCCIÓN



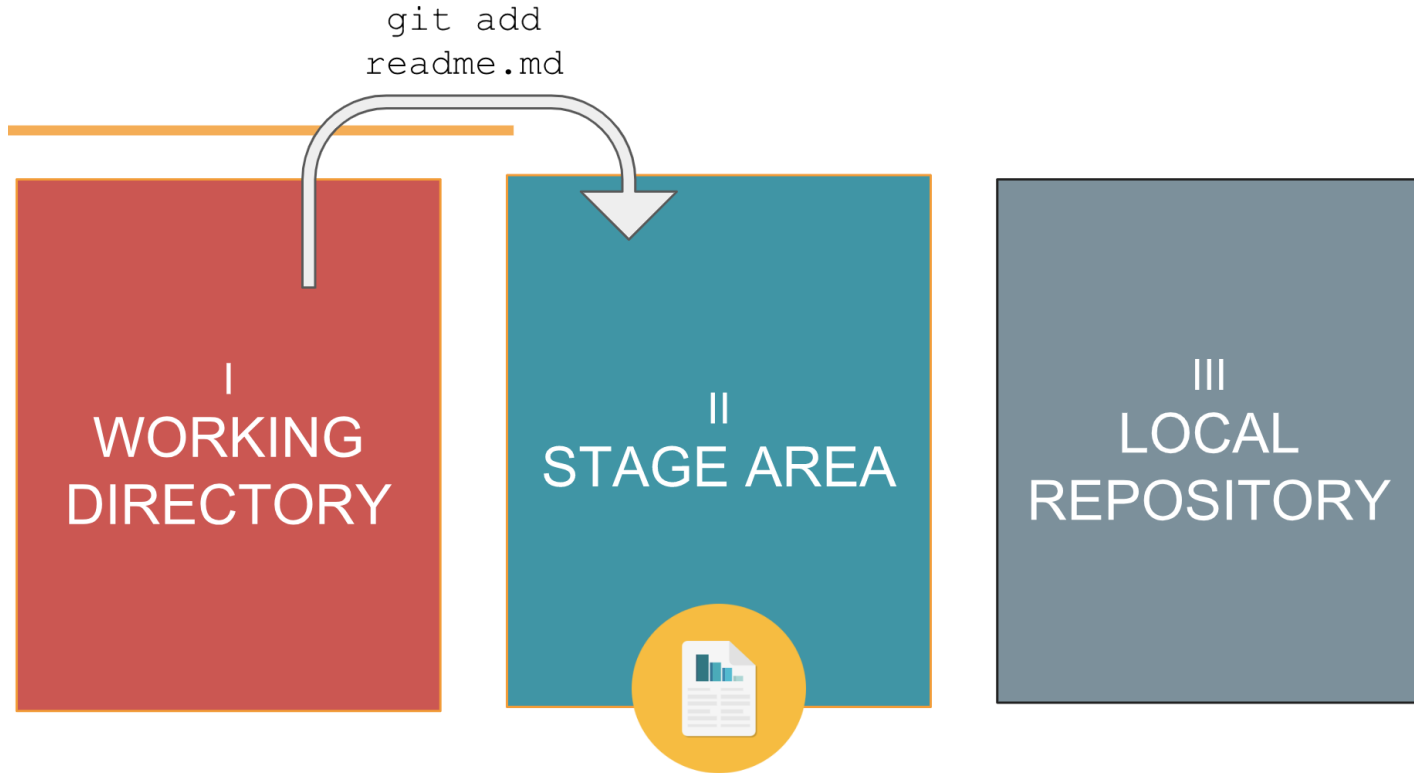
## GIT – OPERACIONES BÁSICAS

**git add <nombre\_del\_archivo>**

Con esto agregamos los archivos modificados del `working directory` al `staging area`.

```
ubuntu@literary-manta: ~/api X + v
ubuntu@literary-manta:~/aprendiendo-git$ git add index.html
ubuntu@literary-manta:~/aprendiendo-git$
```

# GIT - INTRODUCCIÓN





# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git add <nombre\_del\_archivo>**

Puedes añadir tus archivos uno por uno, todos a la vez e incluso especificar reglas:

*#Añadir todos los archivos*

**git add .**

*# Añadir un archivo concreto*

**git add [filename]**



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

```
git add <nombre_del_archivo>
```

*# Añadir todos los archivos omitiendo los nuevos*

```
git add --all
```

*# Añadir todos los archivos con una extensión específica*

```
git add *.txt
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

```
git add <nombre_del_archivo>
```

*# Añadir todos los archivos dentro de un directorio*

```
git add docs/
```

*# Añadir todos los archivos dentro de un directorio y con una extensión específica*

```
git add docs/*.txt
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

```
git add <nombre_del_archivo>
```

*# Añadir todos los archivos dentro de un directorio y con una extensión específica*

```
git add docs/*.txt
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

`git status`

```
ubuntu@literary-manta:~/aprendiendo-git$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   index.html
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

`git commit -m "mensaje descriptivo"`

Con esto pasamos los archivos del *staging area* al *repository*.

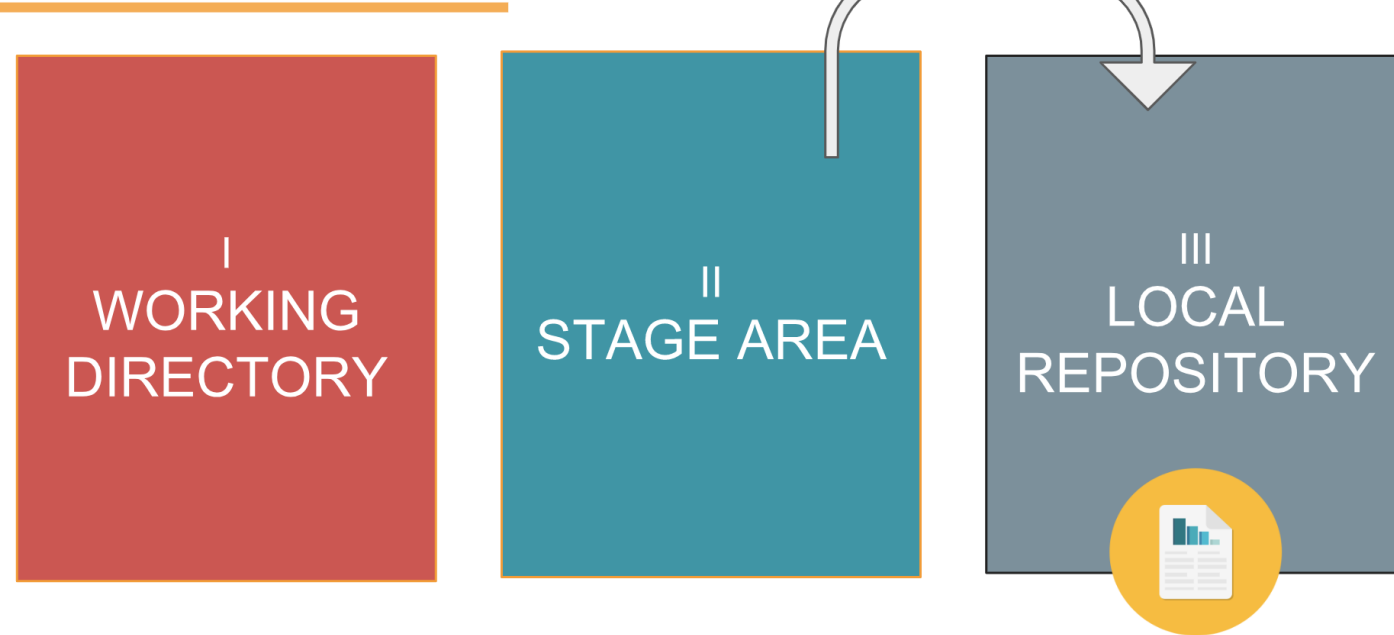
```
ubuntu@literary-manta:~/aprendiendo-git$ git commit -m "Agregando index.html"
[master (root-commit) 2d55618] Agregando index.html
 1 file changed, 7 insertions(+)
 create mode 100644 index.html
ubuntu@literary-manta:~/aprendiendo-git$
```



# GIT - INTRODUCCIÓN



```
git commit -m  
"First commit"
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git commit -m "mensaje descriptivo"**

Podemos añadir los ficheros modificados y hacer el commit en un único paso del siguiente modo:

```
git commit -a -m "Descriptive text for this commit"
```

# ó

```
git commit -am "Descriptive text for this commit"
```





# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

### git log

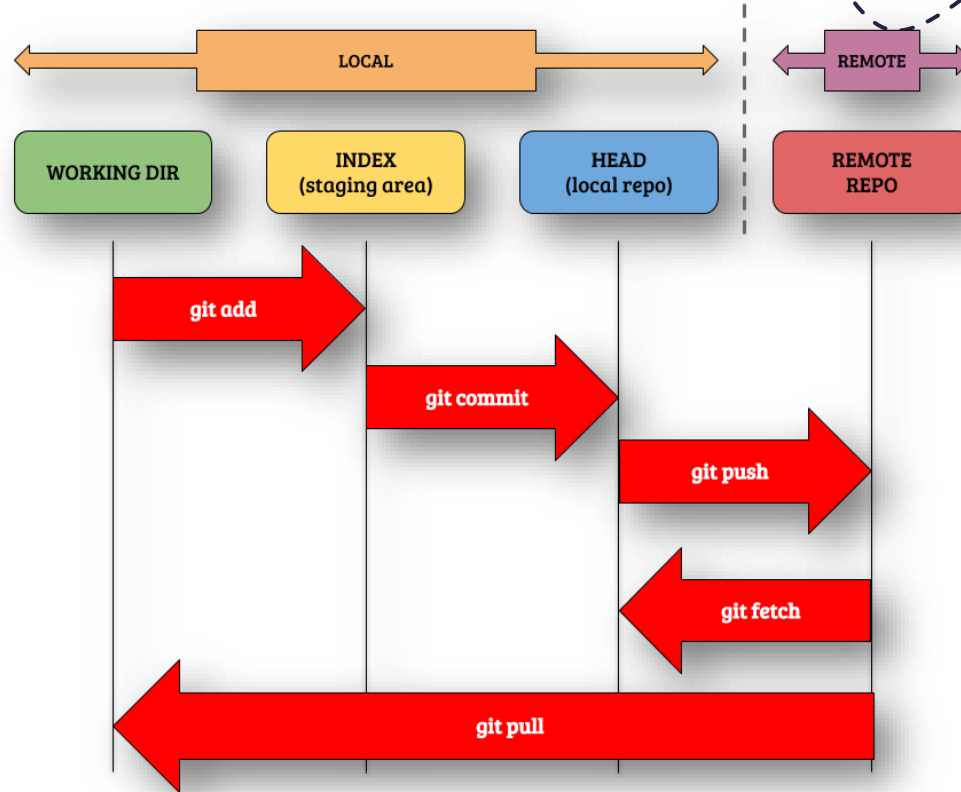
Este comando es muy importante, con este veremos todos los `commits` realizados a lo largo de nuestro proyecto.

```
$ git log
commit 71788626f945c88a1c9ad00b28bc95d936bee63e (HEAD -> master)
Author: carlogilmar <carlogilmar12@gmail.com>
Date:   Fri Feb 22 23:12:26 2019 -0600
```

Agregando archivo readme.md



# GIT - INTRODUCCIÓN



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

`git rm --cached <archivo>`

Elimina archivos después de ser adicionados al index.

```
$ git rm --cached index.html
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

Después del primer commit haz un git log para revisar el estado



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

Luego haz dos modificaciones en el archivo y en cada modificación haz un commit.

Después ejecuta el comando *git log*

Debes observar diferentes puntos de el archivo



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git checkout <id\_del\_commit>**

Con este comando podemos volver a un punto específico de nuestro repositorio (es como viajar en el tiempo)

```
git checkout e902077f157d3adbc43db75c331161bc317d1370
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

### **git reset**

Borra los estados posteriores al commit indicados.

Tipos:

- **Soft:** Borra los commit posteriores pero no borra el directorio de trabajo.
- **Mixed:** Borra los commit posteriores, borra el index área pero no borra el directorio de trabajo.
- **Hard:** Borra todos los commits y el directorio.

# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git reset –soft <id>**

Hacer un *reset soft* al 2do commit que tenemos

```
git reset --soft e902077f157d3adbc43db75c331161bc317d1370
```





# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git reset --soft <id>**

Hacer un *reset soft* al 2do commit que tenemos.

Luego hacer un *git log* y se observa que el ultimo commit fue borrado y el código no fue modificado.

```
git reset --soft e902077f157d3adbc43db75c331161bc317d1370
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git reset --hard<id>**

Hacer un **reset hard** al 1er commit que tenemos.

Luego hacer un **git log** y se observa ha quedado solo el primer commit y que el código fue modificado.

```
git reset --hard abf1eae2848482ae129e479186958219f7a6cf91
```



# GIT - INTRODUCCIÓN



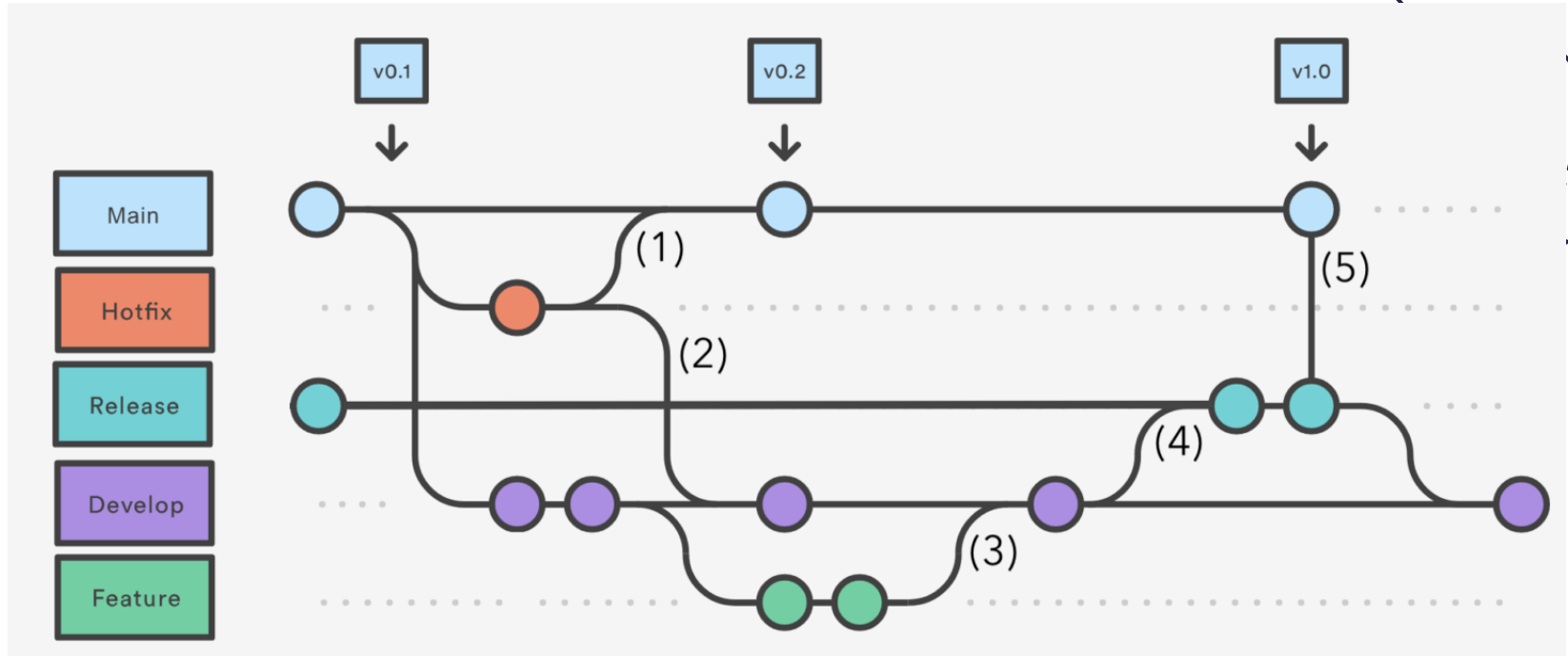
## GIT – OPERACIONES BÁSICAS

### `git branch [-a]`

De esta manera vemos todas las ramas existentes. Por defecto contamos solo con la rama ***master*** que es la principal de nuestro repositorio.



# GIT - INTRODUCCIÓN



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git branch <nombre\_rama>**

Crea una nueva rama con el nombre que se especifica.

```
$ git branch pruebas
```

```
* master  
  pruebas
```

```
git checkout pruebas  
Switched to branch 'pruebas'
```



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git branch <nombre\_rama>**

Luego de crear las ramas y cambiar a la rama prueba, haga un **git log** para ver que estamos en dicha rama.

Realice unos cambios al proyecto en esta rama, para luego fusionarlos con la master.



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

### **git checkout master**

Este command nos lleva a la rama master y salimos de la rama prueba.

Realice un ***git log*** para comprobar que está en master.



# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

**git merge <rama que quiere fusionar>**

Fusiona la rama actual con la que indica.

```
$ git merge pruebas
```

```
Updating 0f9c66c..e44459e
Fast-forward
 index.html | 31 ++++++
 1 file changed, 31 insertions(+)
```





# GIT - INTRODUCCIÓN



## GIT – OPERACIONES BÁSICAS

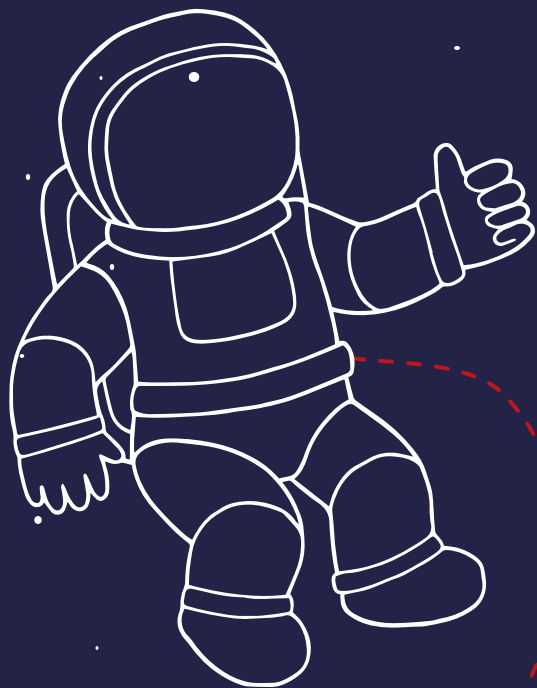
**git merge <rama que quiere fusionar>**

Luego de fusionar haga un **git log** para verificar los commits.

Verifique cuantas ramas tiene con **git branch**



```
git branch
* master
  pruebas
```



# Programa académico CAMPUS

Ciclo 2

