

# Entorno de Programación - Introducción a scm y git

Tecnicatura Universitaria en Inteligencia Artificial

U.N.R.

# Introducción a git

- Sistema de control de versiones (SCM).

# Introducción a git

- Sistema de control de versiones (SCM).
- Esencialmente, rastrea los cambios de nuestros archivos,

- Sistema de control de versiones (SCM).
- Esencialmente, rastrea los cambios de nuestros archivos,  
y permite ir hacia atrás si hace falta.

# Introducción a git

- Creado originalmente por Linus Torvalds en 2005. . .

- Creado originalmente por Linus Torvalds en 2005. . .

en semanas .



- Creado originalmente por Linus Torvalds en 2005. . .

en semanas .



Figure 1: git

# Flexible, jerárquico, distribuido

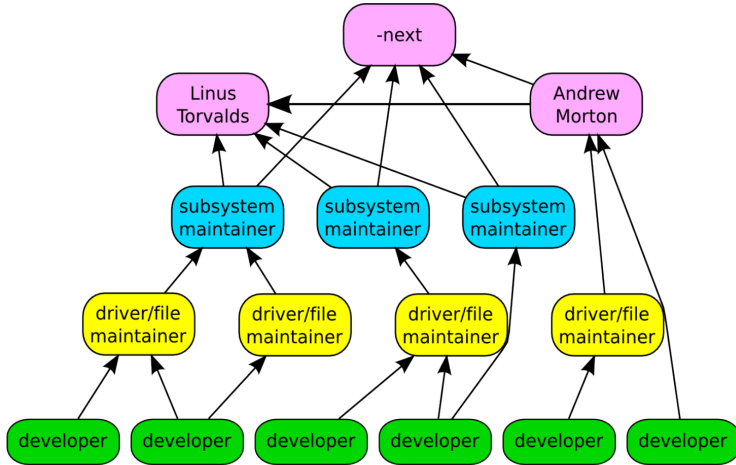


Figure 2: Desarrollo Linux

- Programado en: C, Bourne Shell, Perl

# Introducción a git

- Programado en: C, Bourne Shell, Perl
- Es un proyecto de código abierto

# Introducción a git

- Programado en: C, Bourne Shell, Perl
- Es un proyecto de código abierto
- Una de sus principales características es que es distribuido.

# Introducción a git

- Programado en: C, Bourne Shell, Perl
- Es un proyecto de código abierto
- Una de sus principales características es que es distribuido.
- Cada desarrollador tiene todo el código y toda la historia.

# ¿Cómo arranca un repositorio?

1 inicializando:

# ¿Cómo arranca un repositorio?

1 inicializando:

```
~$ mkdir proyecto/
```



# ¿Cómo arranca un repositorio?

① inicializando:

```
~$ mkdir proyecto/
```

```
~$ cd proyecto/
```

# ¿Cómo arranca un repositorio?

① inicializando:

```
~$ mkdir proyecto/
```

```
~$ cd proyecto/
```

```
~/proyecto$ git init
```

# ¿Cómo arranca un repositorio?

① inicializando:

```
~$ mkdir proyecto/
```

```
~$ cd proyecto/
```

```
~/proyecto$ git init
```

```
Inicializado repositorio Git vacío en /home/aleoncavallo/proyecto/.git/
```

# ¿Cómo arranca un repositorio?

② clonando:

```
$ git clone https://github.com/aleoncavallo/tutorial_bash
```

# ¿Cómo arranca un repositorio?

② clonando:

```
$ git clone https://github.com/aleoncavallo/tutorial_bash
```

```
Clonando en 'tutorial_bash'...
```

```
remote: Enumerating objects: 186, done.
```

```
remote: Counting objects: 100% (35/35), done.
```

```
remote: Compressing objects: 100% (31/31), done.
```

```
remote: Total 186 (delta 18), reused 10 (delta 4), pack-reused 151
```

```
Recibiendo objetos: 100% (186/186), 821.02 KiB | 5.20 MiB/s, listo.
```

```
Resolviendo deltas: 100% (64/64), listo.
```

# Mirando el estado

```
~/tutorial_bash$ git status
```

# Mirando el estado

```
~/tutorial_bash$ git status
```

En la rama master

Tu rama está actualizada con 'origin/master'.

nada para hacer commit, el árbol de trabajo está limpio

# Mirando el estado

```
~/proyecto$ touch otro.sh
```



# Mirando el estado

```
~/proyecto$ touch otro.sh
```

```
~/proyecto$ echo 'echo hola mundo!' > script.sh
```

# Mirando el estado

```
~/proyecto$ touch otro.sh
```

```
~/proyecto$ echo 'echo hola mundo!' > script.sh
```

```
~/proyecto$ git status
```

# Mirando el estado

```
~/proyecto$ touch otro.sh
```

```
~/proyecto$ echo 'echo hola mundo!' > script.sh
```

```
~/proyecto$ git status
```

En la rama master

Tu rama está actualizada con 'origin/master'.

Archivos sin seguimiento:

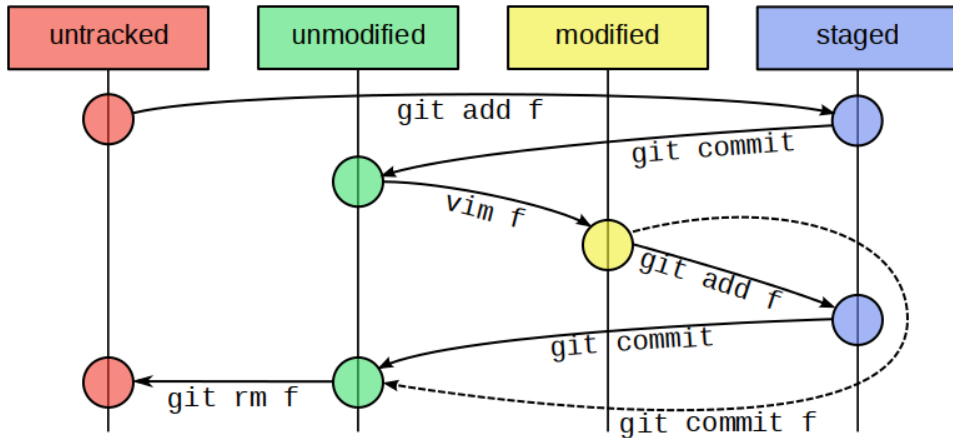
(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)

- otro.sh
- script.sh

no hay nada agregado al commit pero hay archivos sin seguimiento presentes

(usa "git add" para hacerles seguimiento)

# Estados



# Stagin area (escenario)

```
~/proyecto$ git add script.sh
```

# Stagin area (escenario)

```
~/proyecto$ git add script.sh
```

```
~/proyecto$ git status
```

# Stagin area (escenario)

```
~/proyecto$ git add script.sh
```

```
~/proyecto$ git status
```

En la rama master

Tu rama está actualizada con 'origin/master'.

Cambios a ser confirmados:

(usa "git restore --staged <archivo>..." para sacar del área de stage)  
nuevos archivos: script.sh

Archivos sin seguimiento:

(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)  
otro.sh

```
~/proyecto$ git commit -m "Inicio proyecto"
```



# Commit

```
~/proyecto$ git commit -m "Inicio proyecto"
[master (commit-raíz) 4ac0385] Inicio proyecto
1 file changed, 1 insertion(+)
create mode 100644 script.sh
```

# Commit

```
~/proyecto$ git commit -m "Inicio proyecto"
[master (commit-raíz) 4ac0385] Inicio proyecto
 1 file changed, 1 insertion(+)
 create mode 100644 script.sh
~/proyecto$ git status
```

```
~/proyecto$ git commit -m "Inicio proyecto"
```

```
[master (commit-raíz) 4ac0385] Inicio proyecto  
1 file changed, 1 insertion(+)  
create mode 100644 script.sh
```

```
~/proyecto$ git status
```

En la rama master

Archivos sin seguimiento:

(usa "git add <archivo>..." para incluirlo a lo que se será confirmado)  
otro.sh

no hay nada agregado al commit pero hay archivos sin seguimiento presentes  
(usa "git add" para hacerles seguimiento)

# ¿Qué es un commit?

- Snapshot completa del árbol (con optimizaciones de espacio)
- Es un hash criptográfico de:
  - Todos los archivos
  - Mensaje de commit
  - Autor, fecha, etc
  - Commit padre
- Criptográfico = no se puede invertir, ni encontrar colisiones (eficientemente)
- Obviamente... con optimizaciones para no recomputar el hash desde cero cada vez (ver Merkle trees)

```
~/proyecto$ git log -p
commit 4ac03851baed8e79c19ba3c2e3707d0f8477abc8 (HEAD -> master)
Author: Andrea Leon Cavallo <aleoncavallo@gmail.com>
Date:   Tue May 30 17:16:28 2023 -0300
```

Inicio proyecto

```
diff --git a/script.sh b/script.sh
new file mode 100644
index 0000000..34cae35
--- /dev/null
+++ b/script.sh
@@ -0,0 +1 @@
+echo hola mundo!
```

```
~/proyecto$ git show
```

```
commit 06031abc6fd5794e7b5e1bc6b941d0d3984408df (HEAD -> master)
```

```
Author: Andrea Leon Cavallo <aleoncavallo@gmail.com>
```

```
Date: Tue May 30 17:26:00 2023 -0300
```

```
    corregir énfasis en español
```

```
diff --git a/script.sh b/script.sh
```

```
index 34cae35..baaa505 100644
```

```
--- a/script.sh
```

```
+++ b/script.sh
```

```
@@ -1,1 @@
```

```
-echo hola mundo!
```

```
+echo ¡hola mundo!
```

- Commits lo más pequeños posibles (“atómicos”): permite revertir fácilmente
- Mensajes descriptivos: “cambios” vs “Agrego tal funcionalidad”
- Nunca romper el build: permite biseccionar

- Commits lo más pequeños posibles (“atómicos”): permite revertir fácilmente
- Mensajes descriptivos: “cambios” vs “Agrego tal funcionalidad”
- Nunca romper el build: permite biseccionar

`git bisect` Bisect es partir por la mitad y es justamente lo que va a hacer este comando, ir dividiendo toda la pila de commits en dos partes, una parte de la pila contendrá el error y otra parte no.



# Borrar archivos

- `git rm` borra un archivo (y anota el cambio en la staging area). Es lo mismo que hacer `rm` y `git add`.

```
~/proyecto$ git rm script.sh
rm 'script.sh'
~/proyecto$ git status
On branch master
Your branch is up to date with 'origin/master'.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
deleted:
script.sh
$ git commit -m "Borrar main"
[master f068e5f] Borrar main
1 file changed, 1 deletions(-)
delete mode 100644 main.c
```

# Config

```
$ git config --global user.name "Alan Turing"  
$ git config --global user.email "aturing@princeton.edu"
```

Hasta ahora, nada requirió internet.

The screenshot shows the GitLab web interface in a browser. The address bar displays 'gitlab.com'. The left sidebar contains a 'Your work' section with links to 'Projects', 'Groups', 'Issues', 'Merge requests', and 'To-Do List'. The main content area is titled 'Your work > Projects'. Below this, the word 'Projects' is prominently displayed. To the right of 'Projects' are links for 'Explore projects' and a 'New project' button. Below the title, there are filters for 'Yours' (4), 'Starred' (0), and 'Pending deletion'. There is also a search bar labeled 'Filter by name' and two dropdown menus for 'Language' and 'Name'. At the bottom of the filter section, there are tabs for 'All' and 'Personal'.

The screenshot displays the GitLab web interface for creating a new project. The browser address bar shows `gitlab.com/projects/new`. The left sidebar, titled 'Your work', lists various project management tools: Projects (selected), Groups, Issues, Merge requests, To-Do List, Milestones, Snippets, Activity, Workspaces, Environments Dashboard, Operations Dashboard, and Security. The main content area is titled 'Create new project' and offers four distinct options:

- Create blank project**: Create a blank project to store your files, plan your work, and collaborate on code, among other things.
- Create from template**: Create a project pre-populated with the necessary files to get you started quickly.
- Import project**: Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.
- Run CI/CD for external repository**: Connect your external repository to GitLab CI/CD.

The top navigation bar includes a search bar, a 'Actualizar' button, and several utility icons.

Your work / Projects / New project / **Create blank project**



## Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

### Project name

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

### Project URL

### Project slug

Want to organize several dependent projects under the same namespace? [Create a group](#).

### Project deployment target (optional)

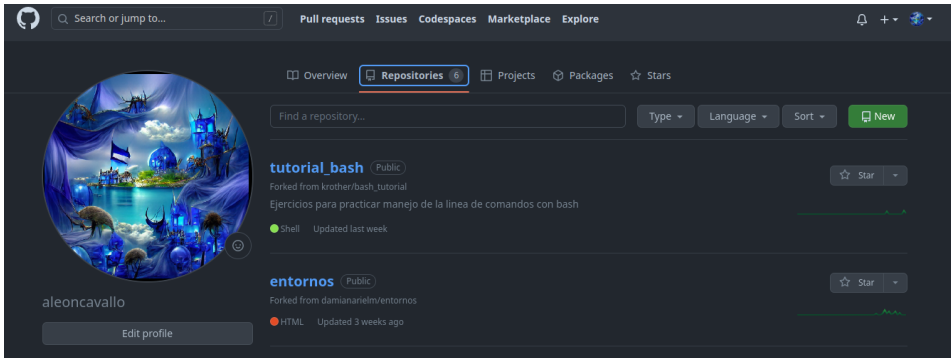
### Visibility Level

☒  Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☐  Public

The project can be accessed without any authentication.



The screenshot shows the GitHub profile of user **aleoncavallo**. The profile picture is a circular image of a fantastical island with a blue sky, a large blue dome, and various creatures. The username **aleoncavallo** is displayed below the picture, with an **Edit profile** button underneath. The navigation bar at the top includes links for Pull requests, Issues, Codespaces, Marketplace, and Explore. The **Repositories** tab is selected, showing a list of repositories. The first repository is **tutorial\_bash**, which is public, forked from **krother/bash\_tutorial**, and described as "Ejercicios para practicar manejo de la linea de comandos con bash". It is a Shell repository updated last week. The second repository is **entornos**, which is public, forked from **damianarielm/entornos**, and described as "Entorno de Programación - Introducción a scm y git". It is an HTML repository updated 3 weeks ago. Both repositories have a star button and a small green progress bar.

aleoncavallo

Edit profile

Repositories

Find a repository...

Type Language Sort New

**tutorial\_bash** Public

Forked from krother/bash\_tutorial

Ejercicios para practicar manejo de la linea de comandos con bash

Shell Updated last week

**entornos** Public

Forked from damianarielm/entornos

HTML Updated 3 weeks ago

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*



aleoncavallo

Repository name \*

/

Great repository names are short and memorable. Need inspiration? How about [expert-engine?](#)

Description (optional)



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Initialize this repository with:



**Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

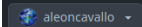
.gitignore template: None



## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*



Repository name \*

/ git\_intro

✔ git\_intro is available.

Great repository names are short and memorable. Need inspiration? How about **expert-engine**?

Description (optional)

Introducción a git



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Initialize this repository with:



**Add a README file**

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

[gitignore templates](#) [None](#)

## Quick setup — if you've done this kind of thing before

or

HTTPS

SSH

`https://github.com/aleoncavallo/git_intro.git`



Get started by creating a new file or uploading an existing file. We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

## ...or create a new repository on the command line

```
echo "# git_intro" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/aleoncavallo/git_intro.git
git push -u origin main
```



```
git remote add origin https://github.com/aleoncavallo/git_intro.git
```

```
git remote add origin https://github.com/aleoncavallo/git_intro.git
```

o desde gitlab

```
git remote add origin git@gitlab.com:aleoncavallo/git_init.git
```

- Push: actualiza el repo remoto desde el local (sólo cambios commiteados)
- Pull: actualiza el repo local desde el remote

Lo más básico ya está

# Revirtiendo cambios

- `git checkout <file>`: revierte cambios locales.
- `git reset`: vacía el staging area.
- `git reset <commit>`: vuelve al commit, sin modificar archivos.
- `git reset --hard \<commit>`: vuelve al commit, descartando todo.

# Branches (Ramas)

- Un branch es un nombre que “apunta” o “sigue” a un commit hash
- `git branch caracteristicaX`: crear y moverse a un branch
- `git checkout <b>`: cambiar de branch
- master suele ser la rama principal



# Merge (Unir ramas)

- Toma dos commits y une sus cambios en uno.

# Merge (Unir ramas)

- Toma dos commits y une sus cambios en uno.
- En branch source `git merge <commit>`.

# Merge (Unir ramas)

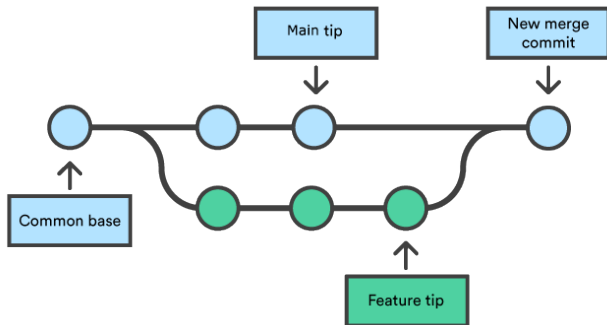
- Toma dos commits y une sus cambios en uno.
- En branch source `git merge <commit>`.

Pull tiene merge implícito, push sólo permite “fast-forwards”.

# Merge (Unir ramas)

- Toma dos commits y une sus cambios en uno.
- En branch source `git merge <commit>`.

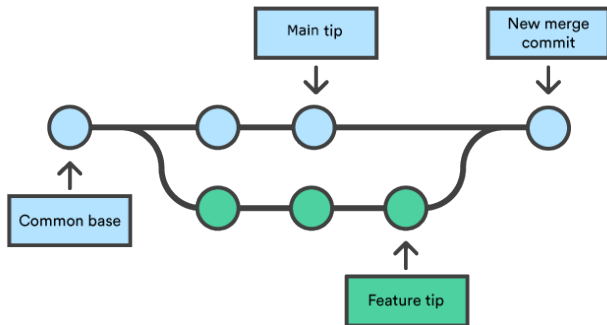
Pull tiene merge implícito, push sólo permite “fast-forwards”.



# Merge (Unir ramas)

- Toma dos commits y une sus cambios en uno.
- En branch source `git merge <commit>`.

Pull tiene merge implícito, push sólo permite “fast-forwards”.



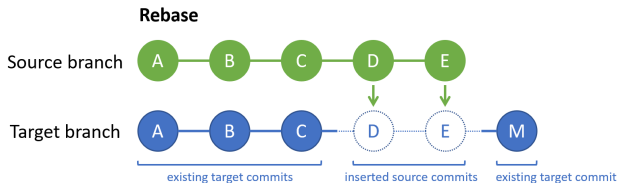
# Merge (Unir ramas)

- Posiblemente haya que corregir conflictos

# Merge (Unir ramas)

- Posiblemente haya que corregir conflictos
- Interfaz gráfica: gitg (o `git log --graph`)

- Similar a un merge... pero “reescribe la historia”



- Generalmente sólo se hace en ramas privadas



Usar con mucho cuidado...

- `git clean -dfx`: borra todo lo que no esté trackeado/staged
- `git clean -x`: borra sólo archivos ignorados (suele ser seguro)
- Flag `-n`: no hacer nada, imprimir lo que haría

# Fuentes y links recomendados

- Sitio oficial de git: <https://git-scm.com/>
- Guía de git: <https://rogerdudler.github.io/git-guide/>
- Aprender a hacer reamas: [https://learngitbranching.js.org/?locale=es\\_AR](https://learngitbranching.js.org/?locale=es_AR)