



Git

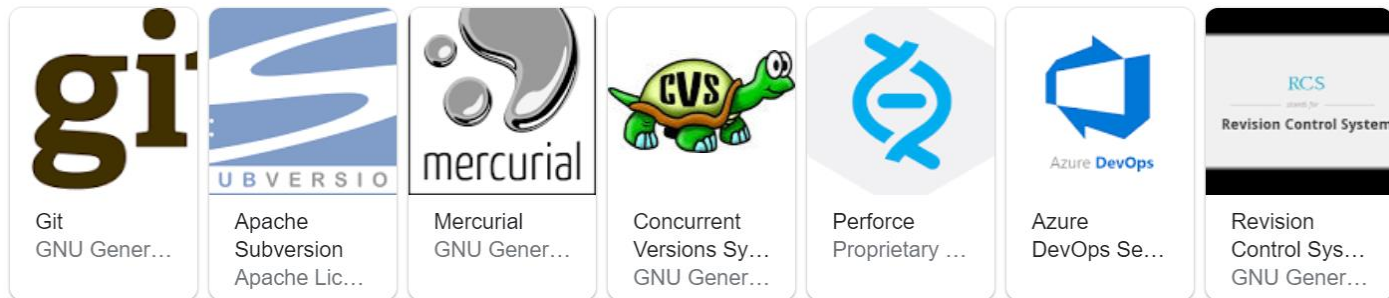
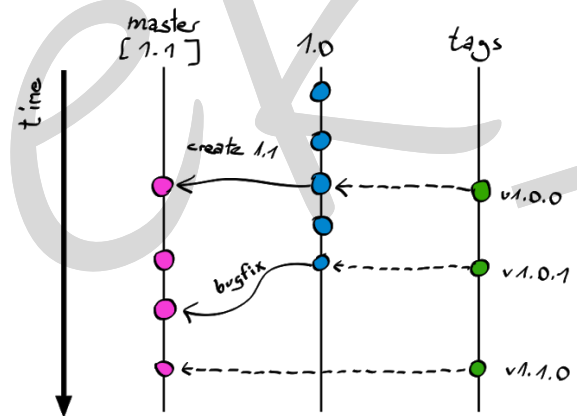
Carolina Caballero

Julio 2021

Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas prácticas
- GitFlow
- GitBash
- GitHub

¿Qué es un manejador de versiones?



¿Qué es un manejador de versiones?

- Version Control System – VCS
- Sirve para tener **control** en los cambios que se realizan en el código o en archivos digitales
 - Control de versiones de forma distribuida
 - Coordina grupos de desarrolladores
 - Podemos ver quién hizo qué y cuándo se hicieron los cambios
 - Ayuda a regresar cambios en algún tiempo determinado
 - Es como una máquina del tiempo
 - Maneja repositorios locales y remotos

Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas prácticas
- GitFlow
- GitBash
- GitHub

Git



git

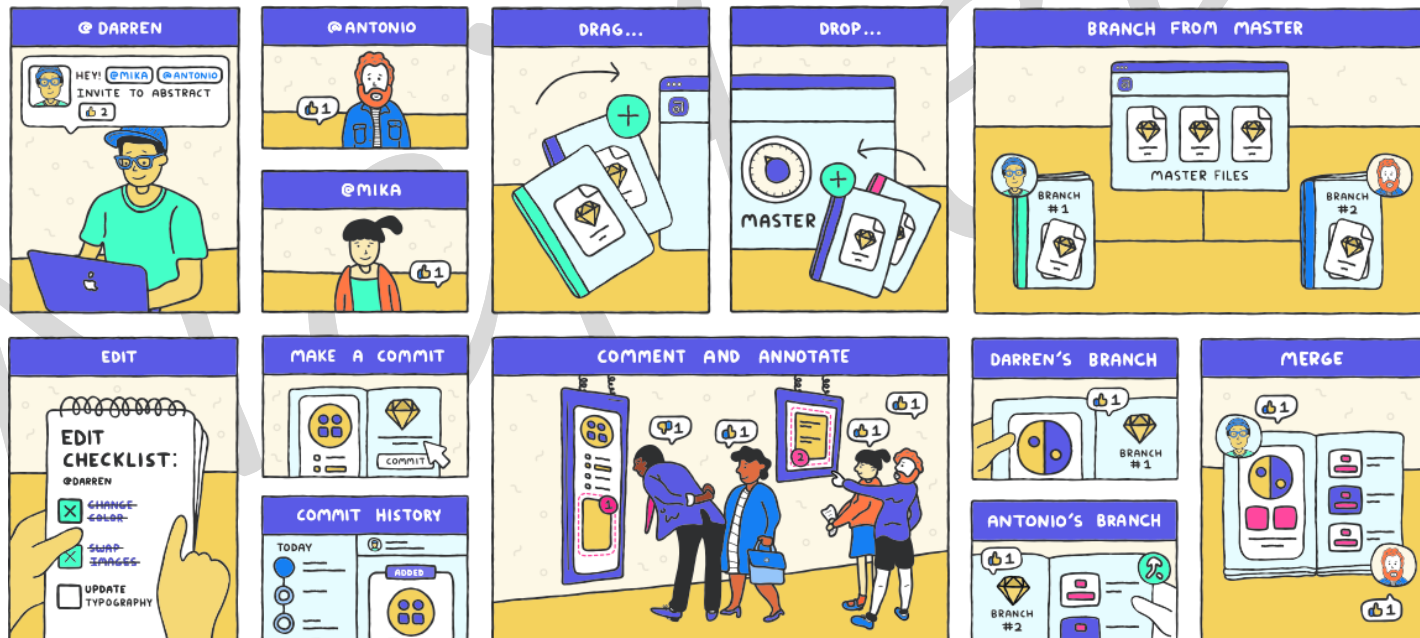
Nace Git

- Creado en 2005, por la comunidad de desarrollo de Linux en particular a Linus Torvalds, el creador de Linux
- Objetivos de este nuevo sistema:
 - Velocidad
 - Diseño sencillo
 - Gran soporte para desarrollo no lineal (miles de ramas paralelas)
 - Completamente distribuido
 - Capaz de manejar grandes proyectos (como el kernel de Linux) eficientemente (velocidad y tamaño de los datos)

Git

- Es una implementación rápida y moderna de **control de versiones** (VCS).
- Nos provee la **historia** de los cambios de contenido
- Facilita trabajo **colaborativo** en cambios de archivos.
- Sirve desde proyectos pequeños y grandes de forma **fácil, rápida** y eficaz.
- Es **gratis** y **open source**.
- Sitio oficial: <https://git-scm.com/>

Cartoon time



Términos mas usados

Término	Significado
Clone	Proceso para clonar o replicar un repositorio alojado en un servidor de versiones.
Stage	Marcado de estado para subir cambios generados en uno o varios archivos de un proyecto en particular.
Commit	Proceso de persistencia de cambios los cuales serán guardados localmente con un mensaje que se especifique según sea el caso.
Push	Proceso de empuje (Subida) de cambios al servidor remoto.

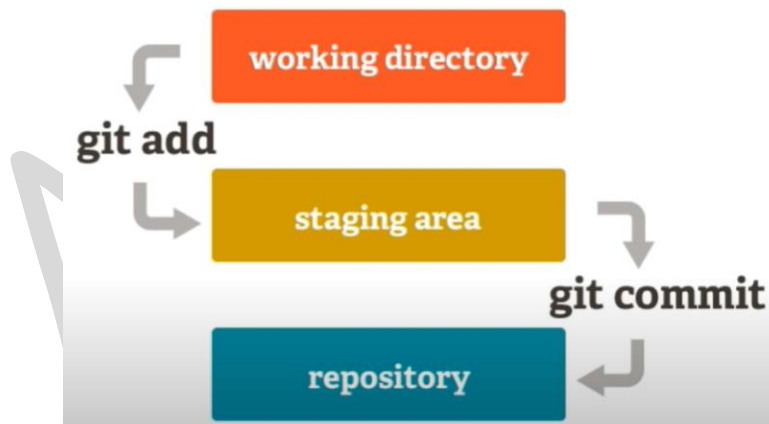
Términos mas usados

Término	Significado
Stash	Marcado de estado temporal para aislar archivos en un contenedor provisional interno para posteriormente restaurarlos si es necesario.
Pull	Proceso de descarga de cambios generados en el servidor remoto.
Merge	Proceso de combinación de versiones, características o ramas, con lo que se aprovecha en mayor medida el trabajo colaborativo.

Términos mas usados

Término	Significado
Branch	Ramas generadas en un proyecto sea para diferentes versiones de un mismo producto con características diferentes o para mejoras continuas del mismo y dar así un desglose coherente del proyecto hasta finalizar cierta actividad de desarrollo.
Pull request (Merge request)	Solicitud de combinación de cambios generados por diferentes ramas las cuales podrán ser aprobados por ciertos usuarios autorizados para realizar este proceso.

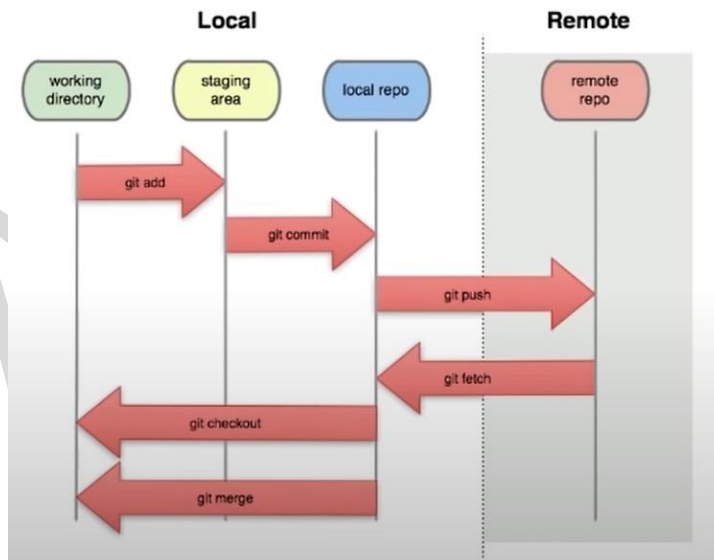
Estados en Git



- Working directory-
donde se trabajas actualmente
tus archivos
- Staging Area – donde se
van mandando los archivos que
se van a mandar al repositorio
- Repository – cuando ya
estas seguro mandas al
repositorio

Flujo de Trabajo en Git

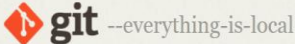
Flujo de trabajo de GIT



- Working directory-
donde se trabajas
actualmente tus archivos
- Staging Area –
donde se van mandando
los archivos que se van a
mandar al repositorio
- Local Repo
- Remote Repo

Instalación de Git


<https://git-scm.com/>



git --everything-is-local

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

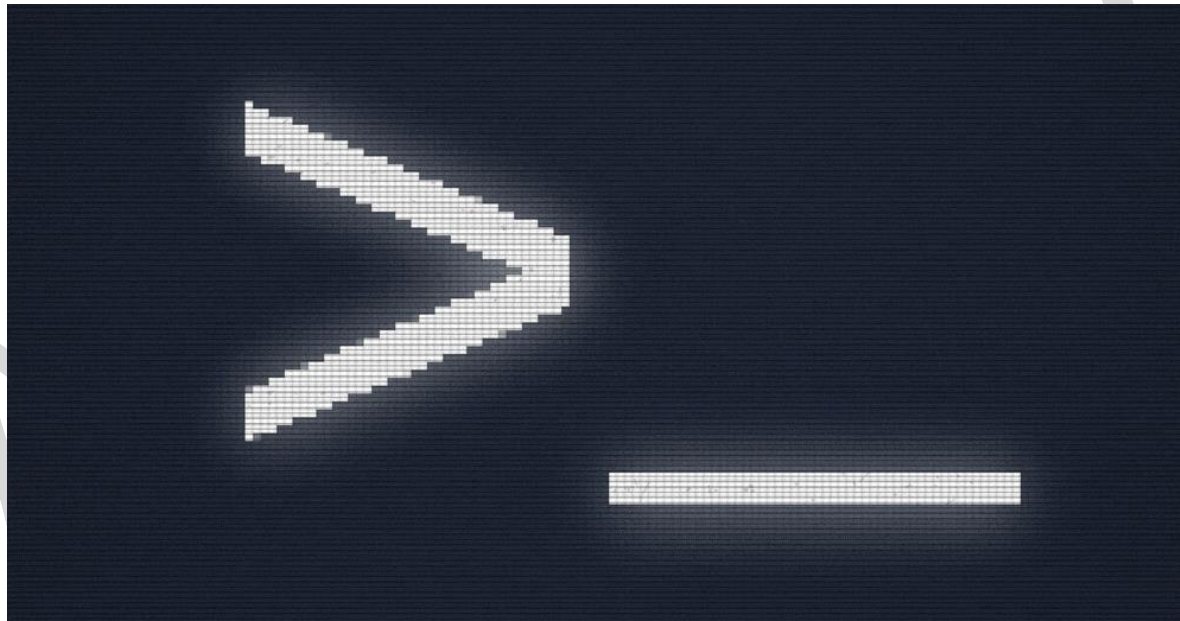
Git is **easy to learn** and has a **tiny footprint** with **lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and multiple workflows.



Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas practicas
- GitFlow
- GitBash
- GitHub

Comandos básicos Git



```
accordion.js index.html main3.css notes.txt readme.txt  
history.txt main.css myScript.js readme.md style.css  
Mario@Mobile-PC MINGW64 /e/GitTutorial/myApp (master)  
$ ls -l  
total 15  
-rw-r--r-- 1 Mario 197121 0 Sep 20 10:55 accordion.js  
-rw-r--r-- 1 Mario 197121 10582 Sep 20 11:34 history.txt  
-rw-r--r-- 1 Mario 197121 160 Sep 19 23:38 index.html  
-rw-r--r-- 1 Mario 197121 0 Sep 19 22:11 main.css  
-rw-r--r-- 1 Mario 197121 0 Sep 20 12:34 main3.css  
-rw-r--r-- 1 Mario 197121 0 Sep 19 22:10 myScript.js  
-rw-r--r-- 1 Mario 197121 0 Sep 19 22:11 notes.txt  
-rw-r--r-- 1 Mario 197121 68 Sep 20 12:33 readme.md  
-rw-r--r-- 1 Mario 197121 10 Sep 20 12:59 readme.txt  
-rw-r--r-- 1 Mario 197121 0 Sep 19 22:10 style.css  
Mario@Mobile-PC MINGW64 /e/GitTutorial/myApp (master)  
$ ls -l
```

Comandos básicos Git

git init

git add <file>

git status

git commit

git push

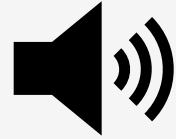
git pull

git clone

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



rek

Cartoon time

Ejercicio Práctico

- Crear un folder llamado: `neitekCursoGit-01`
- Crear un archivo llamado: `index.html`
- Abrir consola cmd
- Iniciar proyecto git
- Agregar archivo `index.html`
- Revisar estatus
- Crear primer commit

Ejercicio práctico

- git init
- se crea directorio oculto .git
- git add index.html
- git status
- git commit -m "primer commit"

```
Initialized empty Git repository in
```

neitekCursoGit-01

Name

.git

index.html

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   index.html
```

```
[master (root-commit) 6ba9809] primer commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
```

Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas prácticas
- GitFlow
- GitBash
- GitHub



8 Buenas prácticas



1. Haz commits sólo de los cambios relacionados

- No corrijas 2 errores en un mismo commit, haz 2 commits separados.
- Hacer pequeños commits facilita la comprensión de los cambios.
- El retroceso es mas sencillo.

2. Haz commits frecuentemente

- Esto ayuda a mantener los commits pequeños.
- Permite compartir tu código con otros con mas frecuencia.
- Es mas fácil integrarse cambios pequeños que grandes.
- La integración de cambios grandes dificulta la resolución de conflictos.

3. No hagas commits de trabajo hecho a medias

- Solo integra tu código cuando esté listo.
- Si estas implementando una funcionalidad grande es mejor dividirla en varias partes para que puedas hacer commits mas pronto.
- Si necesitas pasarte a otra rama, y no quieres perder tus cambios, en vez de dar un commit realiza un **stash**

4. Prueba tu código antes de hacer commit

- No creas en tu instinto de hacer las cosas bien, prueba bien antes de subir tu commit.
- Puede ser un gran problema el subir código no probado, ya que va afectar a los demás.

5. Escribe títulos descriptivos

- Cada commit pone un mensaje que describa lo que se hizo, no solo pongas palabras solas, como “cambios”, “correcciones”.
- Que tus mensajes respondan las preguntas:
 - ¿Cuál es el fin del cambio?
 - ¿En qué se diferencia de la implementación anterior?
- Usa el imperativo presente como:
 - “cambiar”, “añadir”, “corregir”

6. El CV no es un sistema de copias de seguridad

- Aunque es una de las bondades de git o de cualquier sistema de control de versiones, no se debe abusar de ello.
- Haz commits cuando realmente tengas cambios
- No es recomendable hacer commits de archivos compilados, solo los fuentes.

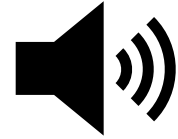
7. Utiliza ramas

- Ramificar es una de las características mas potente de Git.
- De hecho fue uno de los objetivos fundamentales desde el inicio de Git.
- Usa las ramas de forma extensiva: cada nueva función, cada corrección de errores, cada modificación, etc.

8. Define el flujo de Trabajo

- Git permite elegir entre una gran variedad de flujos de trabajo:
 - Ramas de larga duración
 - Ramas temáticas
 - Merge o rebase
 - Git-Flow, etc
- Depende del proyecto, preferencias personales y del equipo de trabajo.

Pónganse de acuerdo!



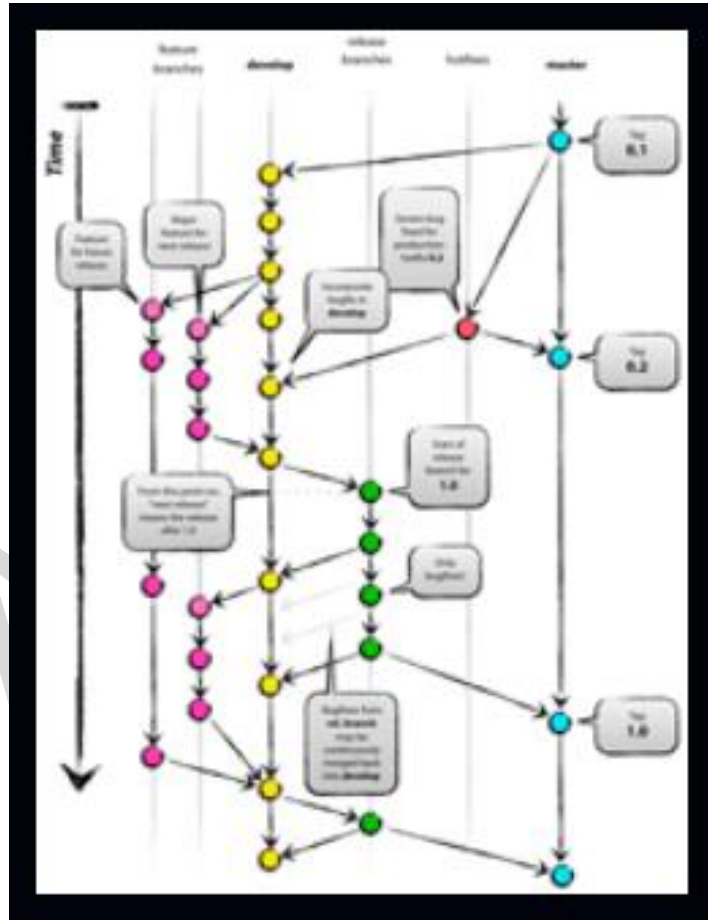
Cartoon time



"No wonder the company is in trouble!"

Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas practicas
- GitFlow
- GitBash
- GitHub



rek -

Git Flow

GitFlow

- Es una metodología de trabajo basada en Git.
- Creada por Vincent Driessen.
- Es ideal para proyectos que tienen un ciclo de lanzamiento programado.
- Permite la paralelización del desarrollo mediante ramas independientes.
- Utiliza básicamente los comandos normales de git (status, add, commit, checkout, merge, pull y push)
- El punto es asignar roles específicos a diferentes ramas y define cómo interactuar

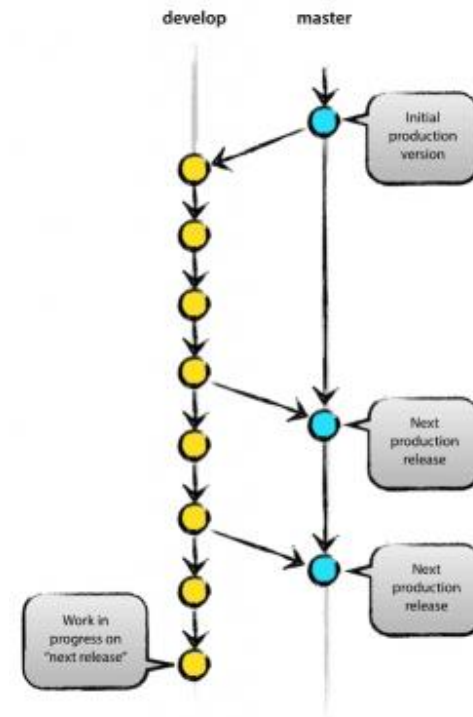
Ramas principales

- Son ramas infinitas
- Rama **master**.

Contiene las versiones estables, las que van a producción

- Rama **develop**.

Contiene el código de desarrollo de la siguiente versión planificada del proyecto



Ramas de apoyo

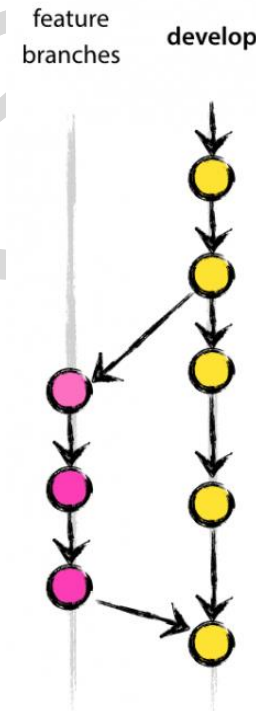
- Son ramas no finitas, serán eliminadas en algún momento
- Ramas:
 - **feature**
 - **release**
 - **hotfix**

Ramas feature

- Se crean en base a la rama **develop**
- Su intención es agregar nueva funcionalidad
- Suelen existir solamente en repositorios locales
- Cuando están terminadas, se manda a develop
- Posteriormente se eliminan estas ramas
- Convención de nombre:

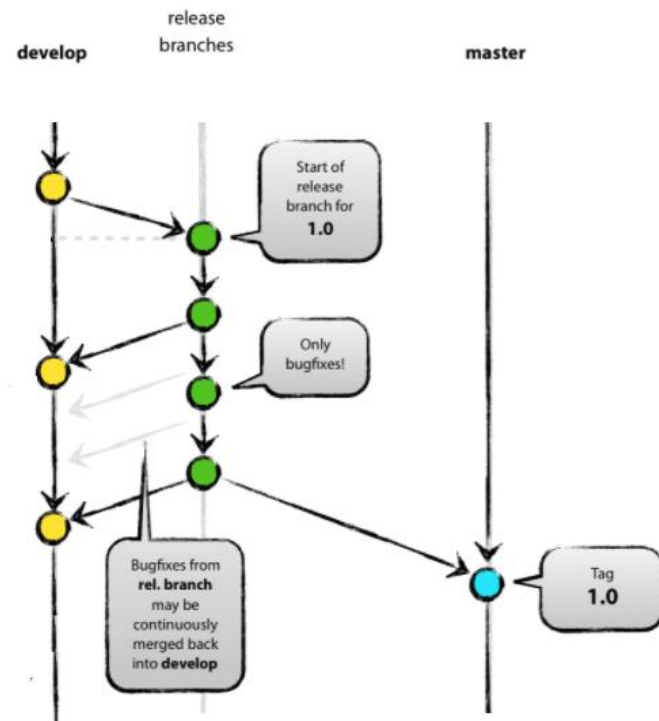
feature/*

Ejemplo: **feature/pet_detail**



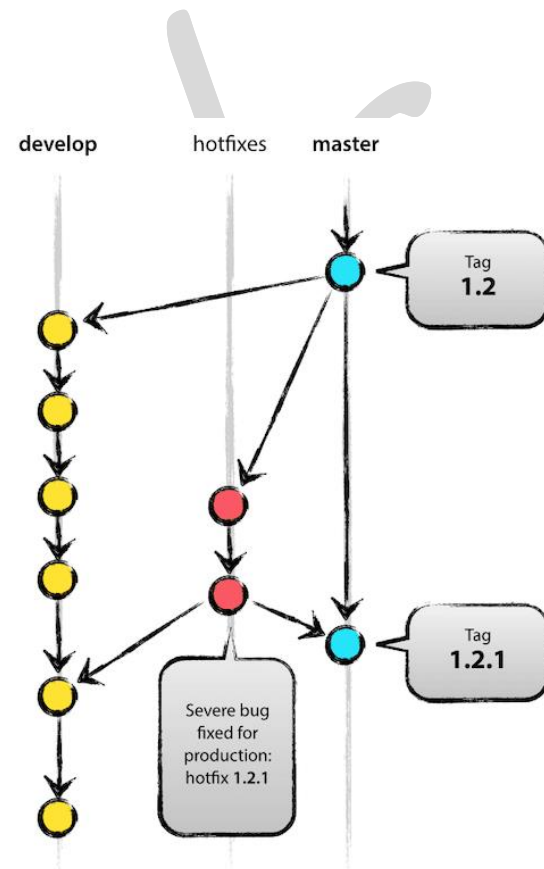
Ramas release

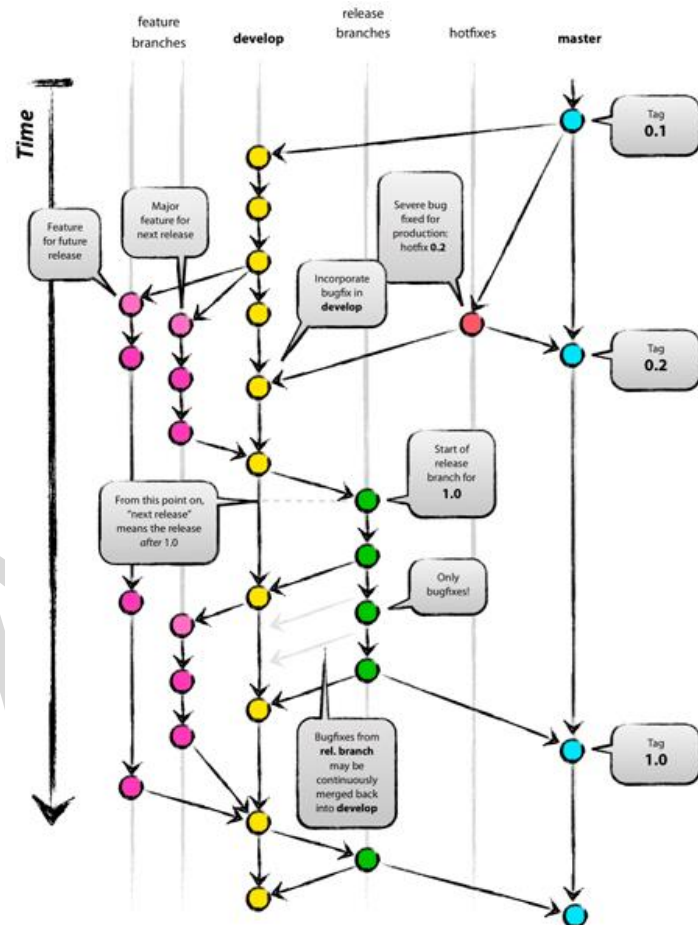
- Se crean en base a la rama **develop**
- Contiene el código que se va a liberar a producción próximamente.
- Cuando está finalizada, se manda a develop y a master
- Rama master se agrega tag (del release)
- Posteriormente se elimina esta rama
- Convención de nombre:
release-<numVersion>
Ejemplo: **release-1.2**



Ramas hotfix

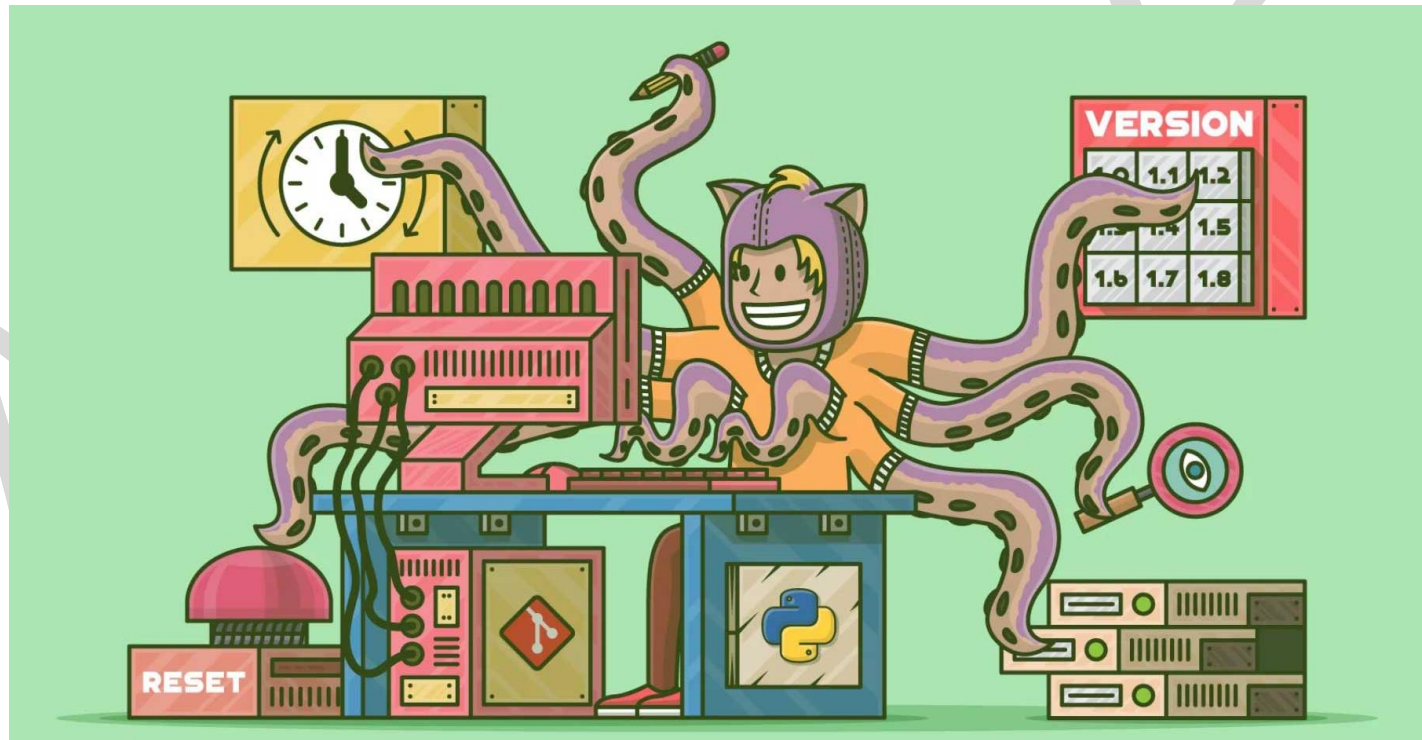
- Surgen de la rama **master**
- Contiene una versión de producción con un error que debe de arreglarse urgentemente.
- Cuando está finalizada, se manda a develop y a master
- Rama master se agrega tag (del hotfix)
- Posteriormente se elimina esta rama
- Convención de nombre:
hotfix-<numRevision>
Ejemplo: **hotfix-1.2.1**



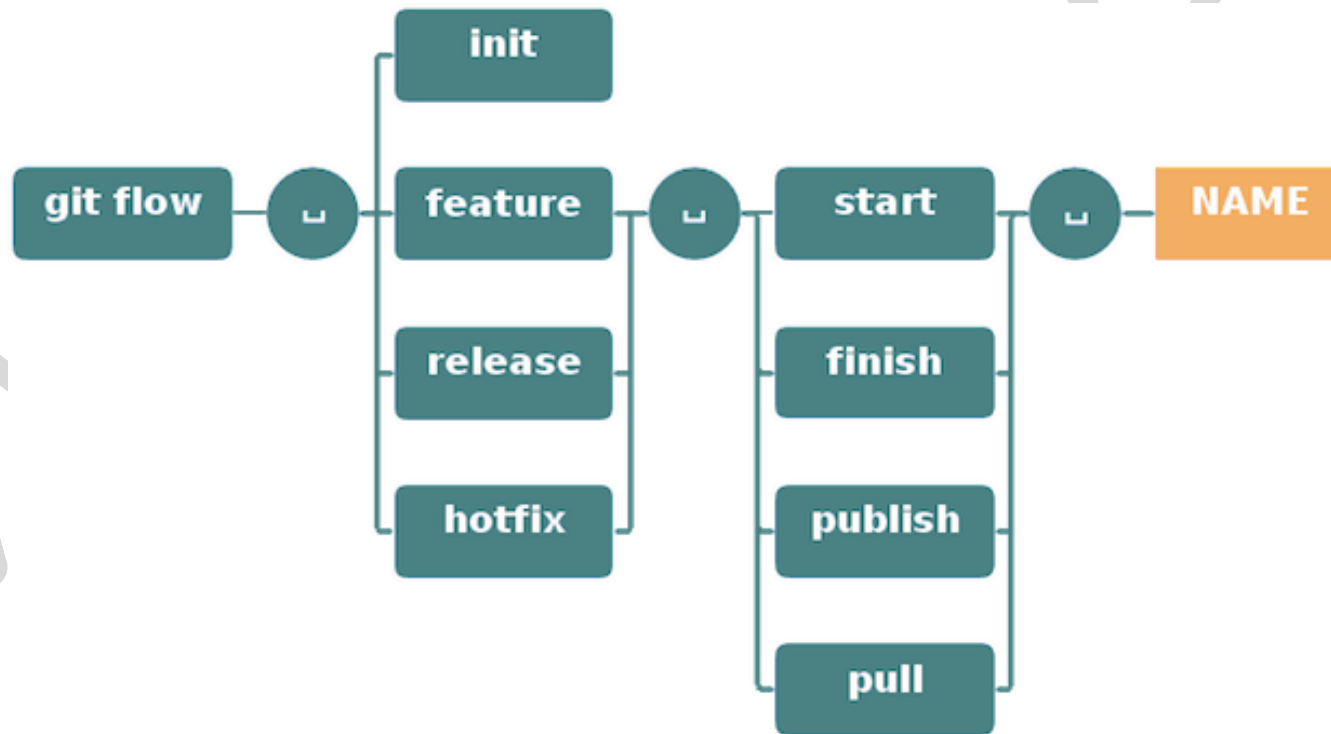


GitFlow

Cartoon time

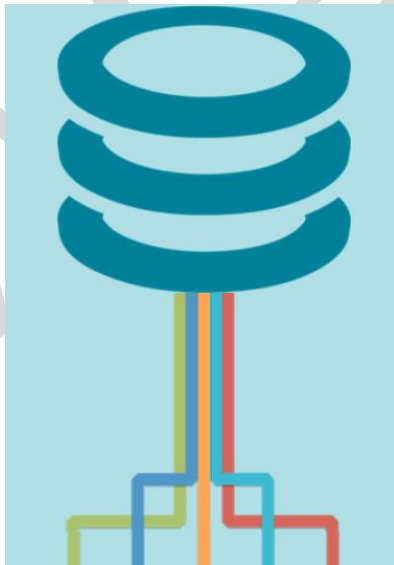


Comandos básicos GitFlow



Iniciar GitFlow

git flow init



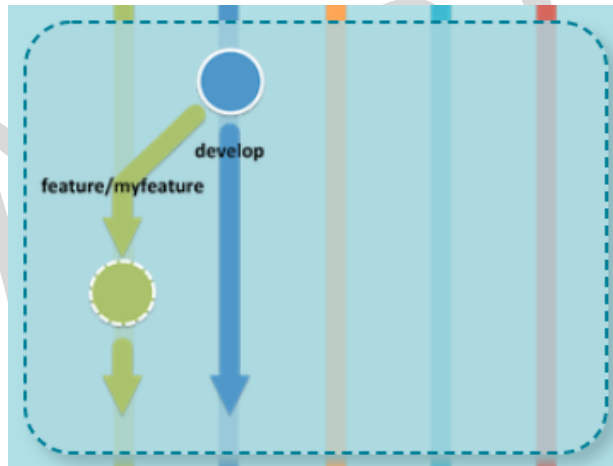
```
Which branch should be used for bringing forth production releases?  
- master  
Branch name for production releases: [master]  
Branch name for "next release" development: [develop]  
  
How to name your supporting branch prefixes?  
Feature branches? [feature/]  
Bugfix branches? [bugfix/]  
Release branches? [release/]  
Hotfix branches? [hotfix/]  
Support branches? [support/]  
Version tag prefix? []  
Hooks and filters directory?
```

Ramas:

```
* develop  
master
```

Feature GitFlow

git flow feature start myFeature1



Switched to a new branch 'feature/myfeature1'

Summary of actions:

- A new branch 'feature/myfeature1' was created, based on 'develop'
- You are now on branch 'feature/myfeature1'

Now, start committing on your feature. When done, use:

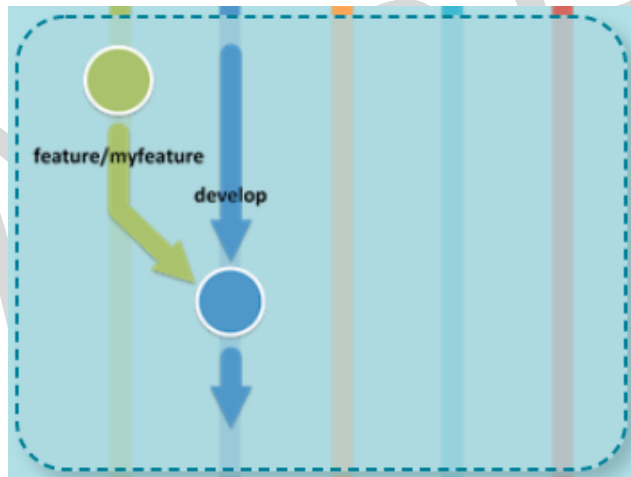
```
git flow feature finish myfeature1
```

Ramas:

```
develop
* feature/myfeature1
master
```

Feature GitFlow

git flow feature finish myFeature1



```
Switched to branch 'develop'  
Already up to date.  
Deleted branch feature/myfeature1 (was 6ba9809).
```

Summary of actions:

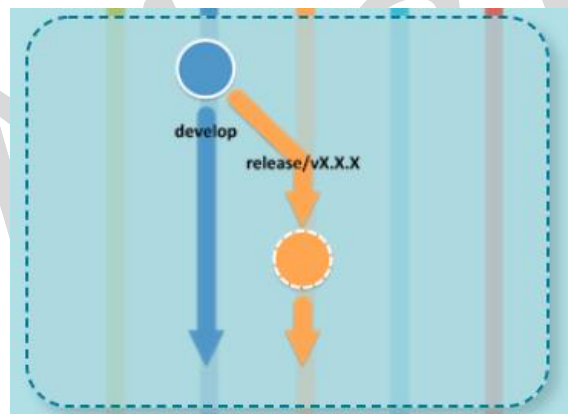
- The feature branch 'feature/myfeature1' was merged into 'develop'
- Feature branch 'feature/myfeature1' has been locally deleted
- You are now on branch 'develop'

Ramas:

```
* develop  
master
```

Release GitFlow

git flow release start release-0.1



Switched to a new branch 'release/release-0.1'

Summary of actions:

- A new branch 'release/release-0.1' was created, based on 'develop'
- You are now on branch 'release/release-0.1'

Follow-up actions:

- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

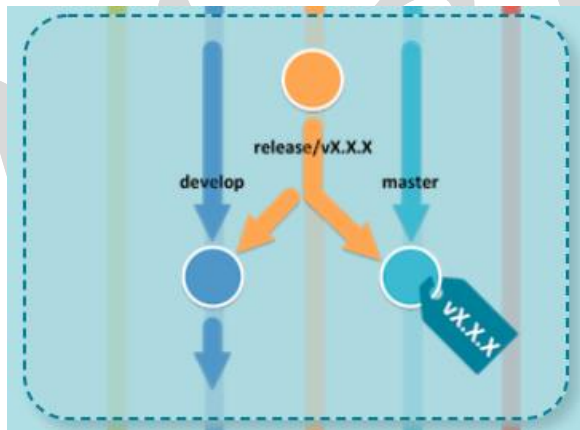
```
git flow release finish 'release-0.1'
```

Ramas:

```
develop
master
* release/release-0.1
```

Release GitFlow

git flow release finish release-0.1



```
Switched to branch 'master'
Merge made by the 'recursive' strategy.
 login.html | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 login.html
Already on 'master'
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
 login.html | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
Deleted branch release/release-0.1 (was d057c60).

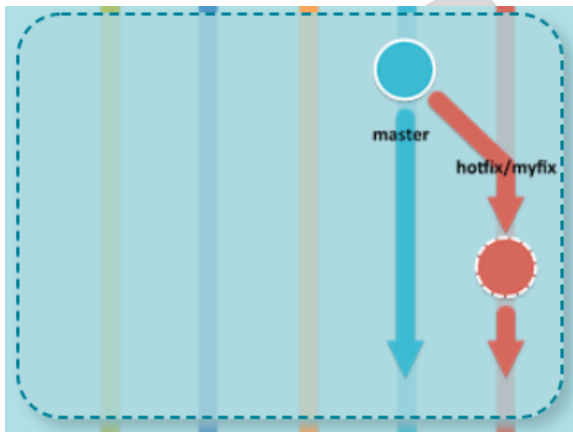
Summary of actions:
- Release branch 'release/release-0.1' has been merged into 'master'
- The release was tagged 'release-0.1'
- Release tag 'release-0.1' has been back-merged into 'develop'
- Release branch 'release/release-0.1' has been locally deleted
- You are now on branch 'develop'
```

Ramas:

```
* develop
  master
```


HotFix GitFlow

git flow hotfix start hotfix-0.1.1



Switched to a new branch 'hotfix/hotfix-0.1.1'

Summary of actions:

- A new branch 'hotfix/hotfix-0.1.1' was created, based on 'master'
- You are now on branch 'hotfix/hotfix-0.1.1'

Follow-up actions:

- Start committing your hot fixes
- Bump the version number now!
- When done, run:

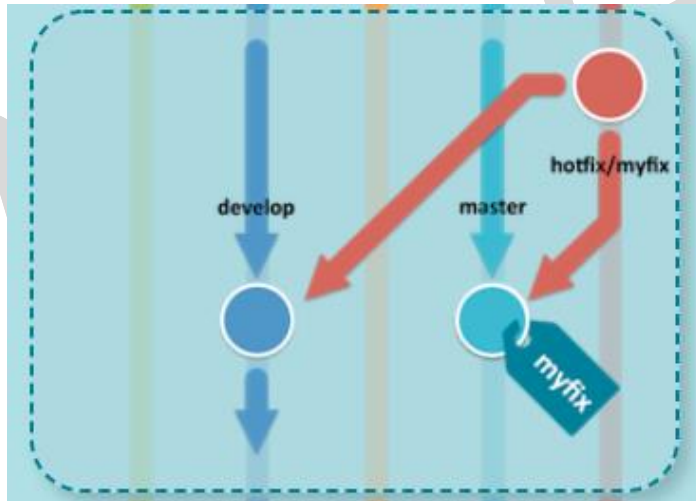
```
git flow hotfix finish 'hotfix-0.1.1'
```

Ramas:

```
develop
* hotfix/hotfix-0.1.1
master
```

Hotfix GitFlow

git flow hotfix finish hotfix-0.1.1



```
Switched to branch 'master'
Merge made by the 'recursive' strategy.
 login.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
 login.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Deleted branch hotfix/hotfix-0.1.1 (was dcfc344).
```

Summary of actions:

- Hotfix branch 'hotfix/hotfix-0.1.1' has been merged into 'master'
- The hotfix was tagged 'hotfix-0.1.1'
- Hotfix tag 'hotfix-0.1.1' has been back-merged into 'develop'
- Hotfix branch 'hotfix/hotfix-0.1.1' has been locally deleted
- You are now on branch 'develop'

Ramas:

```
* develop
master
```

Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas practicas
- GitFlow
- GitBash
- GitHub

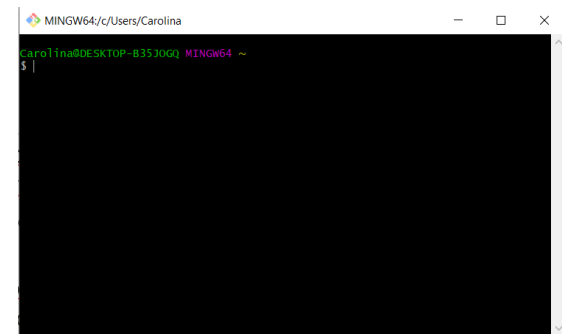
Git Bash



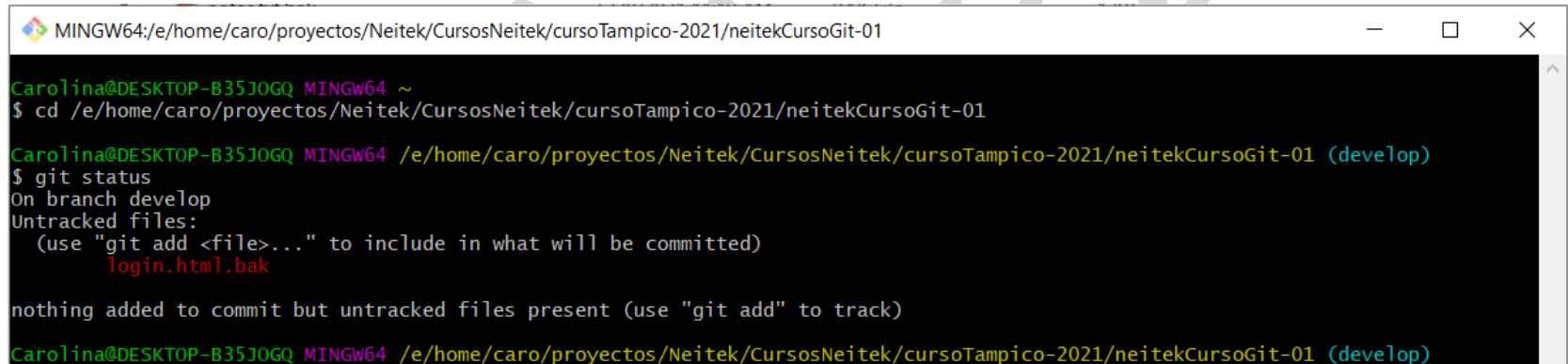
git
bash

Git Bash

- Es una aplicación de Windows, que provee un emulador de línea de comandos para Git.
- Del acrónimo **B**ourne **A**gain **S**hell.
- Se instala como parte de la instalación de Git para Windows.



Git Bash



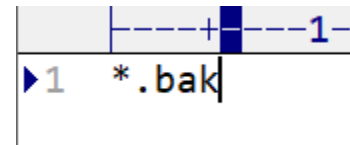
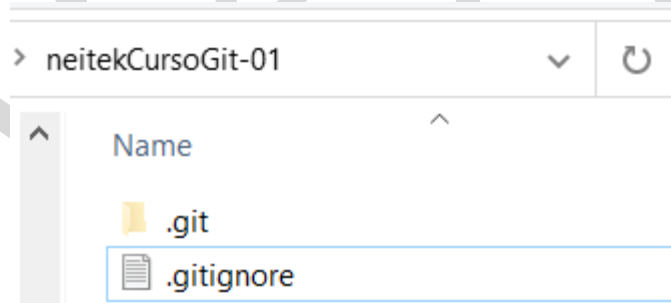
```
MINGW64:/e/home/caro/proyectos/Neitek/CursosNeitek/cursoTampico-2021/neitekCursoGit-01
Carolina@DESKTOP-B35J0GQ MINGW64 ~
$ cd /e/home/caro/proyectos/Neitek/CursosNeitek/cursoTampico-2021/neitekCursoGit-01

Carolina@DESKTOP-B35J0GQ MINGW64 /e/home/caro/proyectos/Neitek/CursosNeitek/cursoTampico-2021/neitekCursoGit-01 (develop)
$ git status
On branch develop
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    login.html.bak

nothing added to commit but untracked files present (use "git add" to track)
Carolina@DESKTOP-B35J0GQ MINGW64 /e/home/caro/proyectos/Neitek/CursosNeitek/cursoTampico-2021/neitekCursoGit-01 (develop)
```

.gitignore

- Es un archivo de texto plano para decirle a git que no tome en cuenta ciertos tipos de archivos.



Agenda

- ¿Qué es un manejador de versiones?
- Git
- Comandos básicos git
- Buenas practicas
- GitFlow
- GitBash
- GitHub

GitHub



GitHub

GitHub

- GitHub es un manejador de Repositorios Remotos
- Se manejan los archivos en la nube
- Provee una manera de compartir tu código con los demás
- Existe muchos otros de este tipo



HELIX TEAMHUB



GITLAB



GITHUB



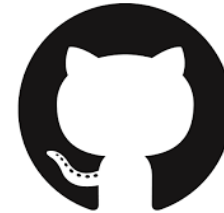
BITBUCKET

Git ≠ GitHub



- Es un sistema.
- Instalado localmente.
- Es una herramienta de línea de comandos.
- Herramienta para manejo de versiones de los archivos de repositorio Git

- Git es la herramienta



- Es un servicio.
- Es hospedado en Web.
- Provee una interface gráfica
- Es un espacio para subir una copia del repositorio Git

- GitHub usa Git

Pasos para usar GitHub

- Crear una cuenta

Sign in

Sign up

Welcome to GitHub!
Let's begin the adventure

Enter your email

→ |

Continue

Join GitHub

Create your account

Username *

Email address *

Password *

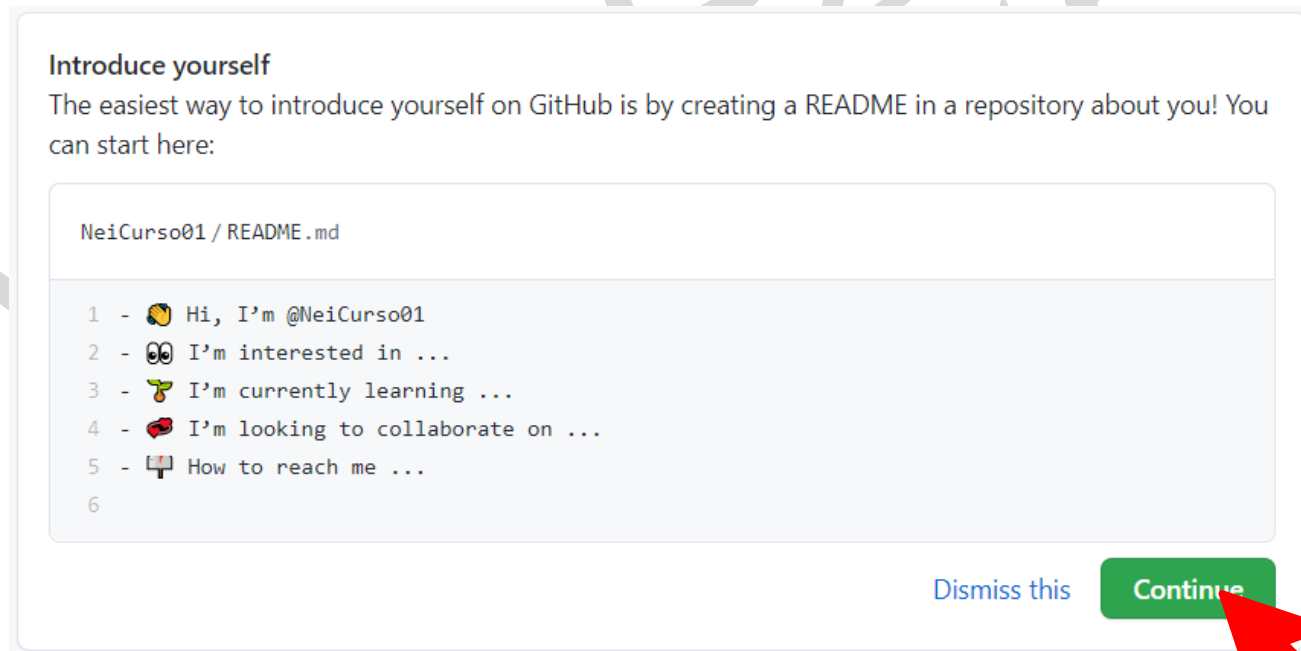
Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter.
[Learn more.](#)

Email preferences

☐ Send me occasional product updates, announcements, and offers.

Pasos para usar GitHub

- Actualizar README.md



Pasos para usar GitHub

● Crear nuevo repositorio

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 *

NeiCurso01 / neiProject01 ✓

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

Description (optional)

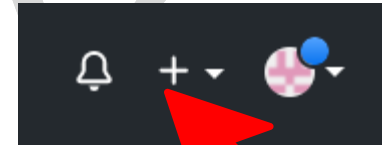
- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)
- ☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
- ☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository



Pasos para usar GitHub

- vincular repositorio local y el remoto

```
git remote add origin  
https://github.com/NeiCurso01/neiProject01.git
```

```
git branch -M master
```

```
git push -u origin master
```

Ramas en Git y GitHub

- Recuerda NO trabajar en master, trabaja con la rama adecuada.

```
git branch
```

```
git branch --all
```

```
git branch develop
```

```
git checkout develop
```

```
git branch --all
```


Cartoon time



FaaS and Furious by Forrest Brazeal

 A CLOUD GURU



Ramas en Git y GitHub



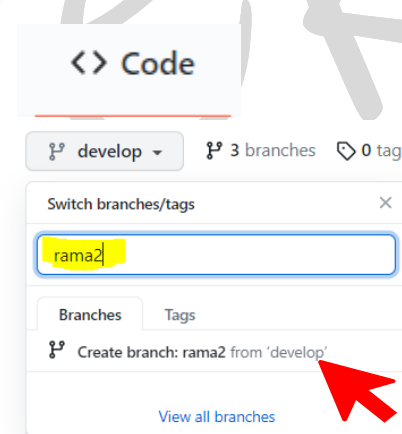
Crear una rama en Git

- git branch rama1
- git checkout rama1

(Trabajar en la rama)

- git push -u origin rama1

Crear una rama en GitHub



- git fetch
- git checkout rama2

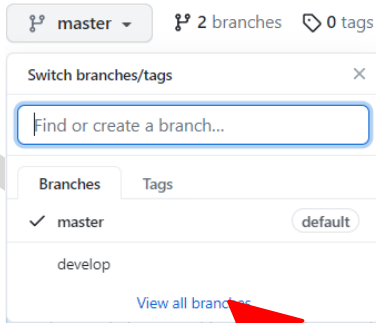
Ramas en Git y GitHub



Borrar una rama en GitHub



<> Code



Borrar una rama en Git



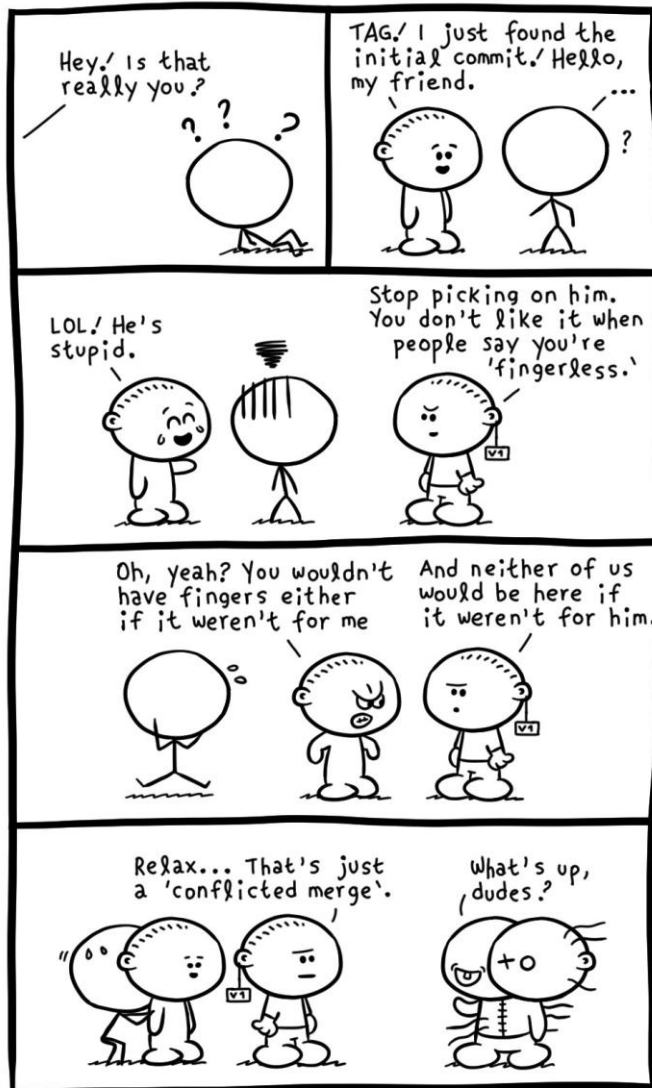
- `git branch --all`
- `git fetch -p`
- `git branch --all`
- `git branch -d rama1`
- `git branch -d rama2`

Ejercicio práctico 1

- Con 2 usuarios, crear 2 ramas para trabajar por separado, subir sus cambios y realizar un PullRequest con reviewer.
 - feature/person_detail
 - feature/employee_detail
- Actualizar rama Develop.

Ejercicio práctico 2

- Con 2 usuarios, crear 1 rama para trabajar sobre esa misma, subir sus cambios y realizar un PullRequest, resolver conflictos.
 - feature/person_list
- Actualizar rama Develop.



Daniel Stori {turnoff.us}



Cartoon time

Para saber mas...

<https://git-scm.com/book/en/v2>

About

Documentation

Reference

Book

Videos

External Links

Downloads

Community

This book is available in [English](#).

Full translation available in

azərbaycan dili,

български език,

Deutsch,

Español,

Français,

Ελληνικά,

日本語,

한국어,

Nederlands,

Русский,

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](#). Print versions of the book are available on [Amazon.com](#).



2nd Edition (2014)

1. Getting Started

- 1.1 About Version Control
- 1.2 A Short History of Git
- 1.3 What is Git?
- 1.4 The Command Line
- 1.5 Installing Git
- 1.6 First-Time Git Setup
- 1.7 Getting Help
- 1.8 Summary

2. Git Basics

- 2.1 Getting a Git Repository
- 2.2 Recording Changes to the Repository
- 2.3 Viewing the Commit History
- 2.4 Undoing Things

Download Ebook



git commit -m "Git course ends"

