

Microsoft Azure Devops (MS-TFS)

Por: Carlos Carreño
ccarreno@cienciadedatos.es

Octubre, 2020



Unidad 1 Introducción a Azure Devops

- Introducción a Git
- Comandos Git
- GitFlow
- Azure Repos
- Integración con GitHub

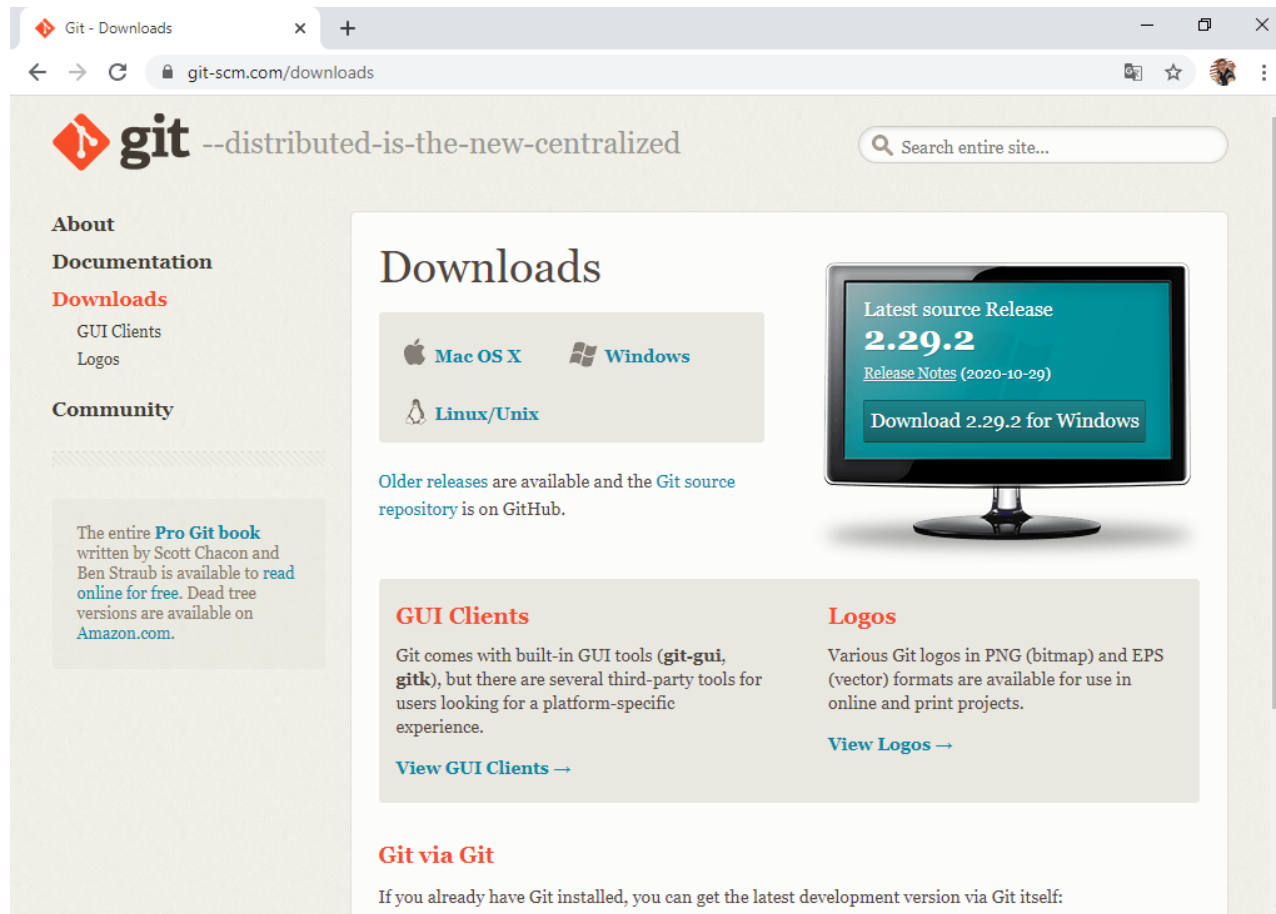


Introducción a Git

- Git es un sistema de control de versiones.

"Sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante".

Instalación de Git

A screenshot of the Git website's 'Downloads' page. The browser window shows the URL 'git-scm.com/downloads'. The page has a sidebar on the left with links for 'About', 'Documentation', 'Downloads' (highlighted), 'GUI Clients', 'Logos', and 'Community'. The main content area is titled 'Downloads' and features a large monitor graphic displaying the 'Latest source Release 2.29.2' with a 'Download 2.29.2 for Windows' button. Below this, there are sections for 'GUI Clients' and 'Logos'. The sidebar also includes a note about the 'Pro Git book' and a link to the 'Git source repository' on GitHub.

Git - Downloads x +

git-scm.com/downloads

git --distributed-is-the-new-centralized

Search entire site...

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Downloads

Older releases are available and the [Git source repository](#) is on GitHub.

Latest source Release
2.29.2
[Release Notes \(2020-10-29\)](#)
[Download 2.29.2 for Windows](#)

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

Git GUIs



Git - GUI Clients

git-scm.com/downloads/guis

Logos

Community

The entire **Pro Git** book written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just follow the instructions.

All

Windows

Mac

Linux

Android

iOS

SourceTree
Platforms: Mac, Windows
Price: Free
License: Proprietary

TortoiseGit
Platforms: Windows
Price: Free
License: GNU GPL

GitHub Desktop
Platforms: Mac, Windows
Price: Free
License: MIT

Git Extensions
Platforms: Linux, Mac, Windows
Price: Free
License: GNU GPL



Comandos Git

- Configuración Inicial

```
git config --global user.name "Nombre que quieras mostrar"
```

```
git config --global user.email "correo@electronico.es"
```

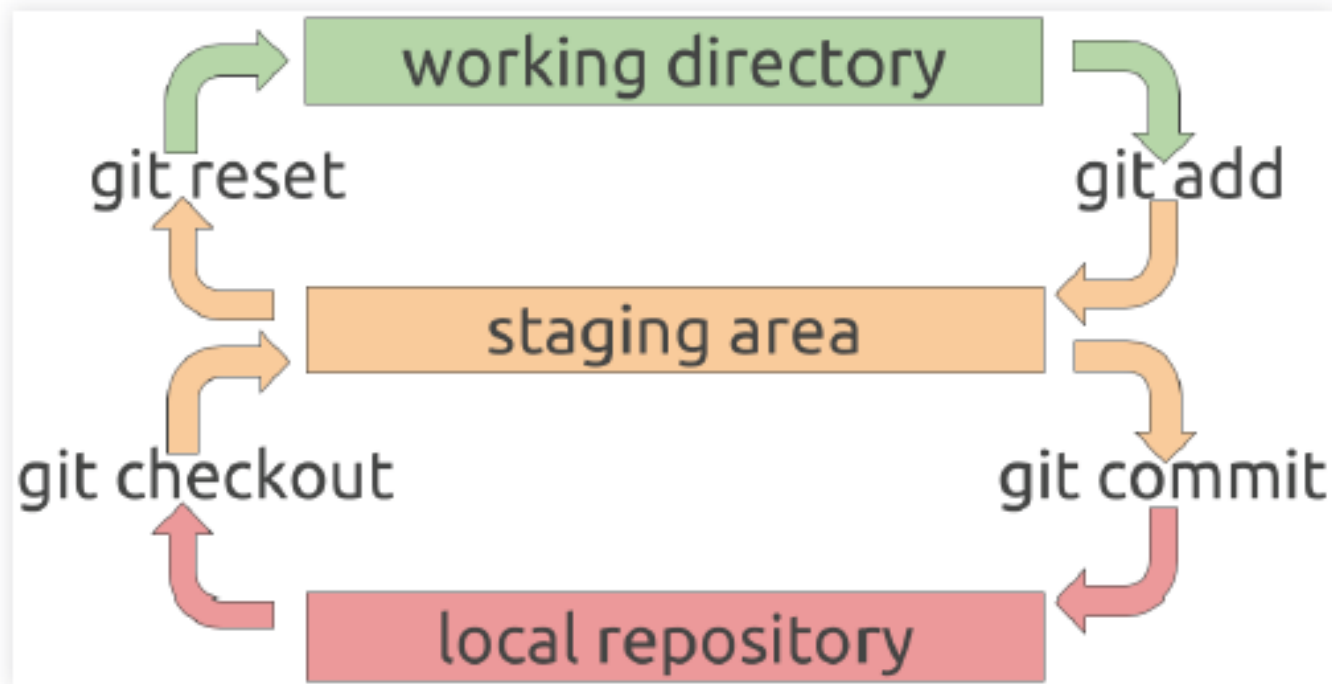
Comandos Git: Inicializar un repositorio



- Crea el **subdirectorio .git** con archivos de git para gestionar el repositorio.

git init

Git: Área de Staging



Staging Area



Git: Estado de los Archivos

- Importante para saber el estado de los archivos.

`git status`

Git: Diferencias



- Podemos ver las diferencias del área de Staging y el área de trabajo

`git diff`



Git: Añadir archivos

- Podemos **añadir** los cambios de un archivo (o varios) al área de Staging (desde el área de trabajo).

```
git add nombre-del-fichero
```

```
git add *.extension
```

```
git add -A
```



Git: Borrar archivos

- Podemos **borrar archivos** del área de staging (también lo borrará del área de trabajo)

```
git rm nombre-del-fichero
```



Git: Mover o renombrar archivos

- Podemos **mover/renombrar archivos** en el área de staging (también lo hará en el área de trabajo)

```
git mv antiguo-nombre-del-fichero nuevo-nombre-del-fichero
```



Git: Resetear archivos

- Para **resetear** los cambios de un archivo (o varios) al área de trabajo (desde el área de staging).

```
git reset nombre-del-fichero
```



Git: Grabar los cambios

- Para **grabar** los cambios realizados al repositorio (desde el área de staging).

```
git commit -m "mensaje corto descriptivo con los cambios"
```



Git: Deshacer los cambios

- Para **deshacer** los cambios de un archivo (o varios) al área de Staging (desde el repositorio).

git checkout nombre-del-archivo



Git: Listado de cambios

- Para ver el **listado de cambios** realizados en el repositorio.

`git log`



Git: Alias

- Podemos crear alias

```
git config --global alias.log 'log --oneline --decorate --graph --all'
```



Git: Ignorar archivos

- Podemos ignorar archivos añadiéndolos al fichero **.gitignore**.



Git: Creando etiquetas

- Existen etiquetas **ligeras**, y etiquetas **anotadas** (iguales pero éstas con más información)

```
git tag nombre-etiqueta-ligera
```

```
git tag -a nombre-etiqueta-anotada -m "mensaje que acompaña a la etiqueta"
```



Git: Etiquetas tardias

- Se puede crear una etiqueta **conociendo el hash del commit** (verlo con git log).

```
git tag -a nombre-etiqueta-annotada -m "mensaje que acompaña a la etiqueta" hash-del-commit
```



Git: Mostrar un etiqueta

- Podemos **ver información concreta de una etiqueta.**

```
git show nombre-etiqueta
```



Git: Retirar una etiqueta

- No podemos sacar una etiqueta, pero podemos **colocar en nuestro directorio de trabajo una versión que coincida con alguna etiqueta, creando una rama nueva:**

```
git checkout -b nombre-rama nombre-etiqueta
```



Git: Conectar a un repositorio remoto

- Podemos **conectar uno o varios repositorios remotos** a nuestro repositorio.

```
git remote add alias-repositorio-remoto url-repositorio-remoto
```




Git: Renombrar un repositorio remoto

- Podemos **renombrar el alias de un repositorio remoto**.

```
git remote rename antiguo-alias nuevo-alias
```



Git: Desconectarse del repositorio

- Podemos **desconectar un repositorio remoto**.

```
git remote remove alias-repositorio-remoto
```



Git: Mostrar los repositorios remotos

- Podemos **ver los repositorios remotos conectados y los permisos que tenemos.**

`git remote -v`



Git: Descargar cambios remotos

- Podemos **descargar los cambios remotos sin modificar nuestro repositorio local.**

```
git fetch alias-repositorio-remoto
```



Git: Descargar y combinar

- Podemos **descargar y combinar los cambios remotos** con los de tu repositorio local.

```
git pull alias-repositorio-remoto nombre-rama-repositorio-remoto
```



Git: Enviar datos

- Podemos **enviar datos al repositorio remoto** (solo si está up-to-date).

```
git push alias-repositorio-remoto nombre-rama-repositorio-remoto
```



Git: Repos y ramas

- Normalmente:

```
git pull/push origin master
```



Git: Enviar datos

- Si queremos subir los tags:

```
git push --tag origin master
```




Git: Clonar repositorio

- Clonar es como:
 - ☐ hacer un init
 - ☐ luego un remote add
 - ☐ luego un fetch con alias=origin
 - ☐ dejando las ramas remota y local en master

```
git clone url-repositorio-remoto
```

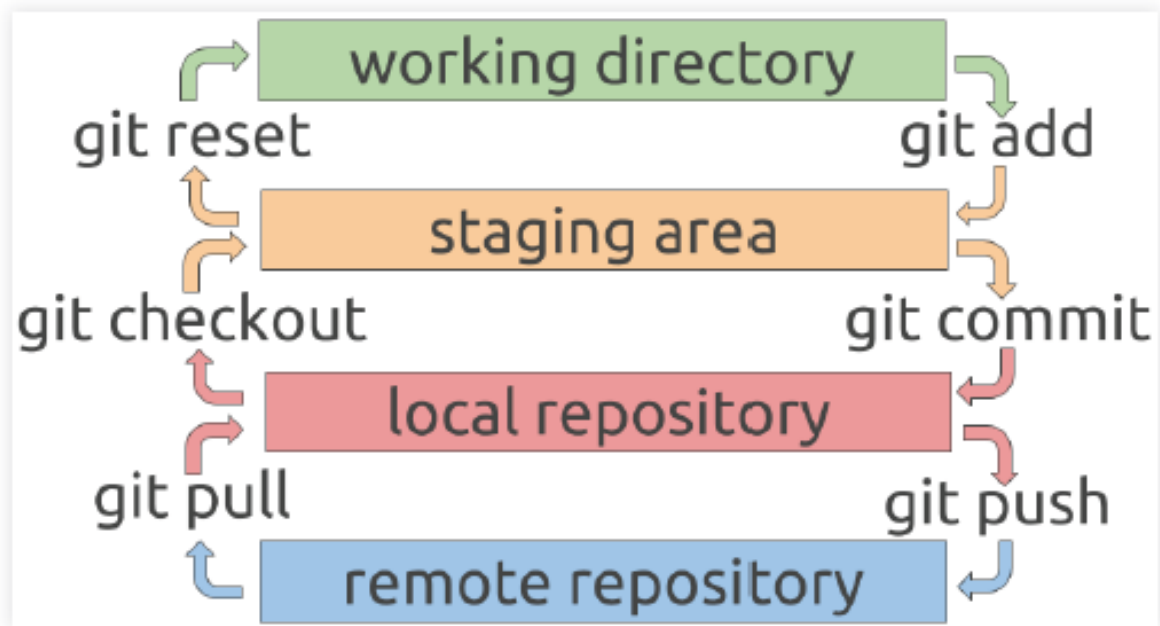
Git: Inspeccionar un repositorio remoto



- Podemos ver **información de un remoto particular, y como están configurados pull y push.**

```
git remote show alias-repositorio-remoto
```

Resumen de Áreas



Resumen áreas GIT



Git: Crear una rama

- Podemos crear ramas que son **apuntadores que podemos mover por los distintos snapshots**.
- Solo la creamos, no nos situamos en ella.

```
git branch nombre-rama
```



Git: Cambiar una rama

- El HEAD es el apuntador que usa GIT para saber en qué rama estás.
- Cuando cambiamos de rama GIT **cambia el HEAD y los archivos de tu área de trabajo.**

```
git checkout nombre-rama
```



Git: Crear y cambiar de rama

- Podemos **crear y cambiar de rama** con un mismo comando.

```
git checkout -b nombre-rama
```



Git: Ramas y el Head

- Podemos **ver las ramas y donde apunta el HEAD.**

```
git log --oneline --decorate --graph --all
```

```
git branch -v
```



Git: Fusionar ramas

- GIT es **muy potente** con la fusión de ramas.

```
git merge nombre-rama
```




Git: Solucionar Conflictos

- Si al hacer un merge existan conflictos **GIT los apunta en los propios archivos.**

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">please contact us at support@github.com</div>
>>>>>> issue:index.html
```



Git: Eliminar ramas

- Una vez fusionado la rama en el master, **conviene borrarla** (solo nos deja si está fusionada).

```
git branch -d nombre-rama
```



Git: Listado de ramas por estado

- Podemos saber **qué ramas están fusionadas y cuáles no.**

```
git branch --merged
```

```
git branch --no-merged
```



Git: Sincronizar rama remota

- Igual que sincronizamos la rama master remota, podemos **sincronizar otras ramas remotas**.

```
git checkout -b nombre-rama-local alias-repositorio-remoto/nombre-rama-remota
```

```
git checkout --track alias-repositorio-remoto/nombre-rama-remota
```



Git: Asignar rama remota

- Podemos **asignar el área de trabajo a una rama remota**.

```
git checkout -u alias-repositorio-remoto/nombre-rama-remota
```



Git: Listado de todas las ramas

- Podemos listar no solo las ramas locales, sino **también las remotas**.

```
git branch -vv
```



Git: Eliminar rama remota

- Podemos **eliminar las ramas remotas**.

```
git push alias-repositorio-remoto --delete nombre-rama-remota
```

Git Flow

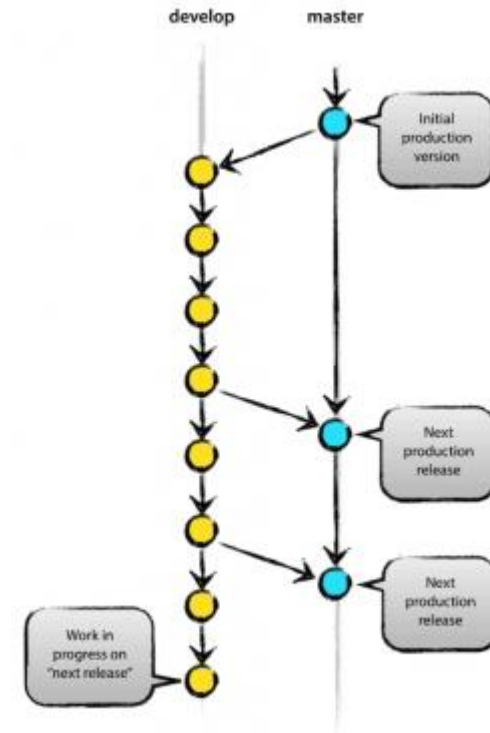


- Git Flow es una metodología de trabajo aplicado al repositorio GIT.
- Vincent Driessen fue el encargado de popularizarlo, definiendo un modelo estricto de ramificación diseñado en torno a los lanzamientos del proyecto.
- Es ideal para proyectos que lleven una planificación de entregas iterativas.
- Permite la paralelización del desarrollo mediante ramas independientes para la preparación, mantenimiento y publicación de versiones del proyecto



Git Flow: Ramas principales

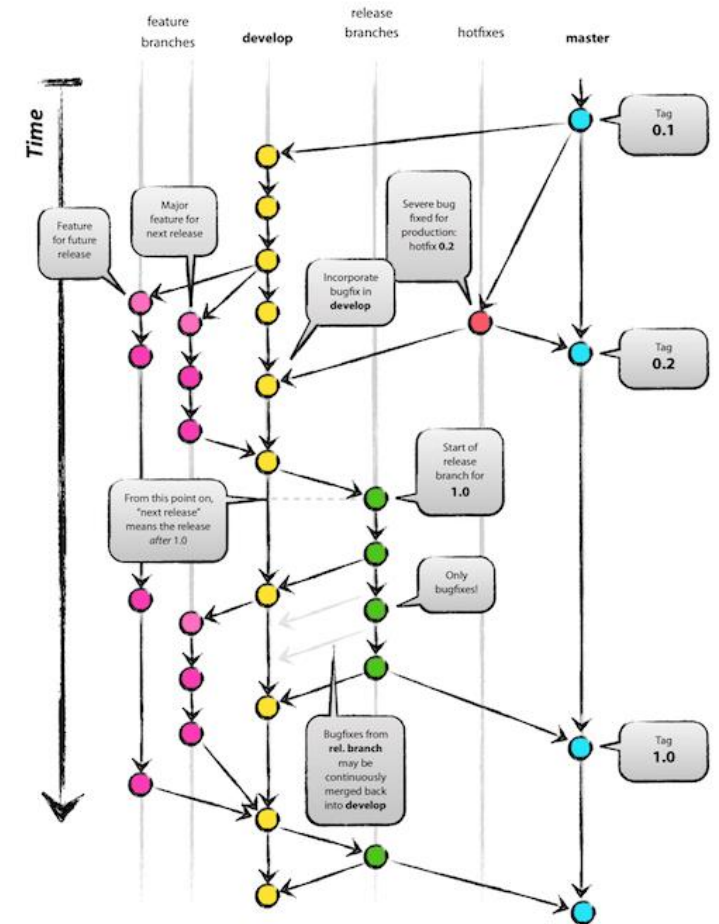
- Todo proyecto, por defecto, debería tener al menos dos ramas infinitas para su desarrollo. Esta metodología define que deben existir dos ramas principales:
 - ❑ **Master:** contiene las versiones estables del proyecto.
 - ❑ **Develop:** Contiene el código de la siguiente version planificada del proyecto.



Git Flow: Ramas de Apoyo



- Junto a las ramas **master** y **develop**, existe un conjunto de ramas de apoyo, su objetivo es el permitir el desarrollo en paralelo entre los miembros del equipo, la resolución de problemas en producción de forma rápida.
- A diferencia de las ramas principales, estas están limitadas en tiempo. Serán eliminadas eventualmente.
 - ☐ feature
 - ☐ release
 - ☐ hotfix





Git Flow: Ramas feature

- Estas ramas tienen que surgir de la rama *develop*. Cada una de estas ramas almacenan código de desarrollo con nuevas características.
- Típicamente existen solamente en los repositorios locales de los desarrolladores y no en el repositorio origen.
- Una vez terminado su desarrollo, se incorporarán nuevamente a la rama *develop*, que contendrá la última versión de código en desarrollo.
- *Convención de nombres*: estas ramas se pueden nombrar de cualquier forma, excepto ***master***, ***develop***, ***release-****, o ***hotfix-****.



Git Flow: Ejemplo, Rama Feature

Crear una rama feature

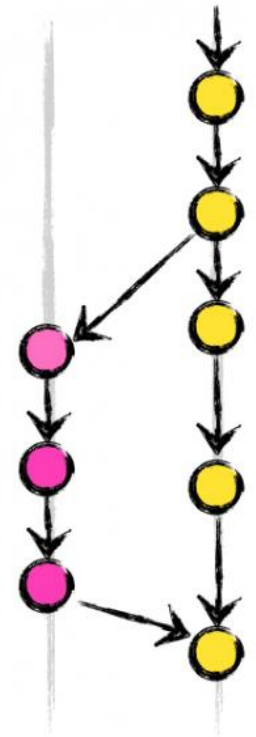
1. `$ git checkout -b feature/myfetaure develop`
2. Switched to a new branch "feature/myfetaure"

Finalizar una rama feature

1. `$ git checkout develop`
2. Switched to branch 'develop'
3. `$ git merge --no-ff feature/myfetaure`
4. Updating eal82a..05e9557
5. (Summary of changes)
6. `$ git branch -d feature/myfetaure`
7. Deleted branch feature/myfetaure (was 05e9557).
8. `$ git push origin develop`

feature
branches

develop





Git Flow: Ramas release

- Como las ramas feature, las ramas release también tienen que surgir de la rama *develop*.
- Contienen el código de la versión que se va a liberar próximamente. Es un paso previo y preparatorio para la versión definitiva de producción. En ella se incluye todo el código de *develop* necesario para el lanzamiento. Puede que contenga algún error pequeño que se debe de arreglar en este momento para no incluirlo en producción.
- Una vez finalizada la rama, esta se debe incluir tanto en la rama *develop* como en la rama *master*.
- *Convención de nombres*: deben de seguir la siguiente convención: **release-***, sustituyendo el * por el número de versión (1.1, 2.3, 4.7, etc)



Git Flow: Ejemplo, rama release

Crear rama release

```
1. $ git checkout -b release-1.2 develop
2. Switched to a new branch "release-1.2"
3. (Hacer las modificaciones necesarias)
4. $ git commit -a -m "Release version 1.2 of the project"
5. [release-1.2 74d9424] Release version 1.2 of the project
6. 1 files changed, 1 insertions(+), 1 deletions(-)
```

Finalizar una rama release

Primero debemos actualizar la rama master.

```
1. $ git checkout master
2. Switched to branch 'master'
3. $ git merge --no-ff release-1.2
4. Merge made by recursive.
5. (Summary of changes)
6. $ git tag -a 1.2
```



...

- A continuación, debemos guardar esos cambios en la rama *develop*.

```
1. $ git checkout develop
2. Switched to branch 'develop'
3. $ git merge --no-ff release-1.2
4. Merge made by recursive.
5. (Summary of changes)
```

- Una vez integrada la rama tanto en *master* como en *develop* eliminaremos la rama en el repositorio local.

```
1. $ git branch -d release-1.2
2. Deleted branch release-1.2 (was ff452fe).
```



Git Flow: Ramas hotfix

- Estas ramas surgen de la rama *master*. Contienen una versión de producción con un error que se desea arreglar urgentemente.
- Una vez arreglado el error, se incluye el contenido de esta rama en las ramas *master* y *develop* para subsanar el error. Además, hay que marcar la versión arreglada de producción con un *tag* en la rama *master*.
- *Convención de nombres*: deben de seguir la siguiente convención: **hotfix-***, sustituyendo el * por el número de la revisión (1.1.5, 2.3.1, 4.7.9, etc)

Git Flow: Ejemplo, rama hotfix



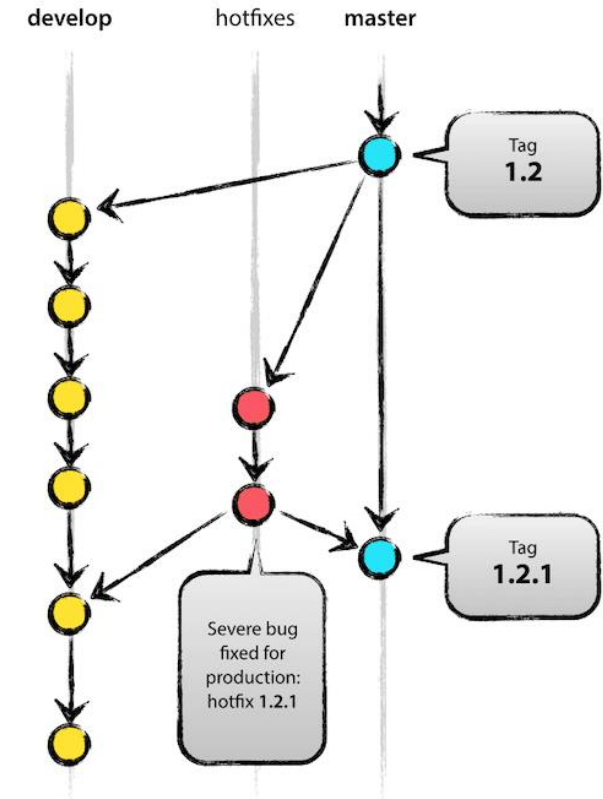
Crear una rama hotfix

1. `$ git checkout -b hotfix-1.2.1 master`
2. Switched to a new branch "hotfix-1.2.1"
3. (Hacer las modificaciones necesarias)
4. `$ git commit -a -m "Bumped version number to 1.2.1"`
5. `[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1`
6. 1 files changed, 1 insertions(+), 1 deletions(-)

Finalizar una rama hotfix

Primero debemos actualizar la rama *master* y etiquetarla.

1. `$ git checkout master`
2. Switched to branch 'master'
3. `$ git merge --no-ff hotfix-1.2.1`
4. Merge made by recursive.
5. (Summary of changes)
6. `$ git tag -a 1.2.1`





...

A continuación debemos incluir el hotfix en *develop* también.

```
1. $ git checkout develop
2. Switched to branch 'develop'
3. $ git merge --no-ff hotfix-1.2.1
4. Merge made by recursive.
5. (Summary of changes)
```

Una vez integrada la rama tanto en *master* como en *develop* eliminaremos la rama en el repositorio local.

```
1. $ git branch -d hotfix-1.2.1
2. Deleted branch hotfix-1.2.1 (was abbe5d6).
```



Herramienta Git Flow

- Tenemos a nuestra disposición una herramienta de línea de comandos que nos ayudará en este proceso, ya que se encarga de realizar todos los pasos intermedios necesarios para gestionar las ramas.

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.18362.1139]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Daddy>git flow
usage: git flow <subcommand>

Available subcommands are:
  init      Initialize a new git repo with support for the branching model.
  feature   Manage your feature branches.
  bugfix    Manage your bugfix branches.
  release   Manage your release branches.
  hotfix    Manage your hotfix branches.
  support    Manage your support branches.
  version   Shows version information.
  config    Manage your git-flow configuration.
  log       Show log deviating from base branch.

Try 'git flow <subcommand> help' for details.

C:\Users\Daddy>
```

Inicialización de metodología Git flow en el repositorio



- Para comenzar a utilizar la metodología Git flow, debemos iniciarla dentro de un repositorio git existente. Para ello ejecutaremos el comando:

```
1. $ git flow init
```



Gestionar features

Crear una nueva *feature* (en el ejemplo *my-feature*):

```
1. $ git flow feature start feature/my-feature
```

Finalizar la rama *feature*:

```
1. $ git flow feature finish feature/my-feature
```

Publicar la rama *feature* en el repositorio remoto:

```
1. $ git flow feature publish feature/my-feature
```

Obtener una rama *feature* del repositorio remoto:

```
1. $ git flow feature pull origin feature/my-feature
```

Seguir de los cambios de la *feature*:

```
1. $ git flow feature track feature/my-feature
```



Gestionar releases

Comenzar una *release* (en el ejemplo *release-1.2*):

```
1. $ git flow release start release-1.2
```

Concluir una *release*:

```
1. $ git flow release finish release-1.2
```

Publicar la *release* en el repositorio remoto:

```
1. $ git flow release publish release-1.2
```

Debemos, también, publicar los *tags* en el repositorio remoto:

```
1. $ git push --tags
```

Seguir los cambios de la *release*:

```
1. $ git flow release track release-1.2
```



Gestionar hotfixes

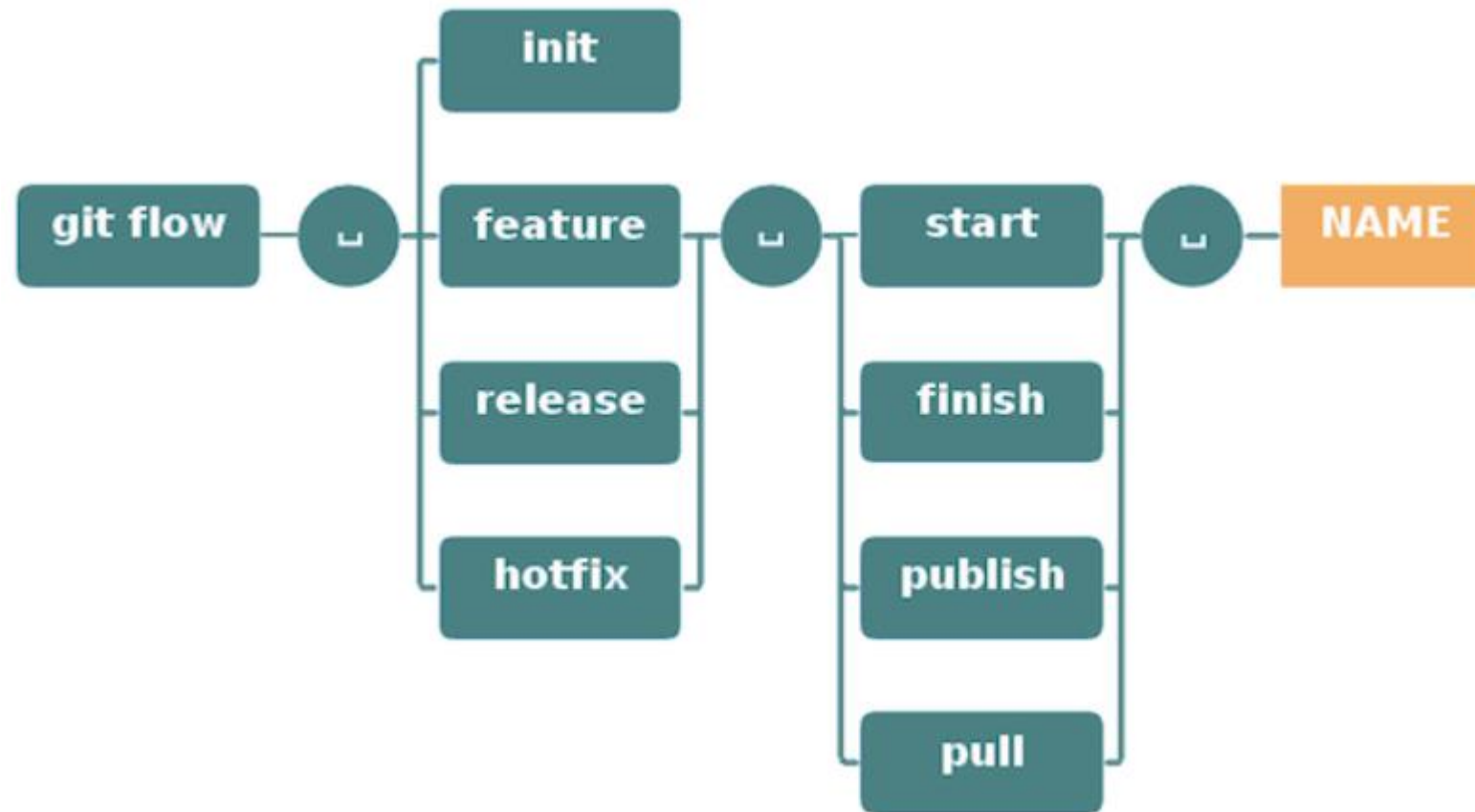
Crear un *hotfix* (en este ejemplo *hotfix-1.2.1*):

```
1. $ git flow hotfix start hotfix-1.2.1
```

Concluir un *hotfix*:

```
1. $ git flow hotfix finish hotfix-1.2.1
```

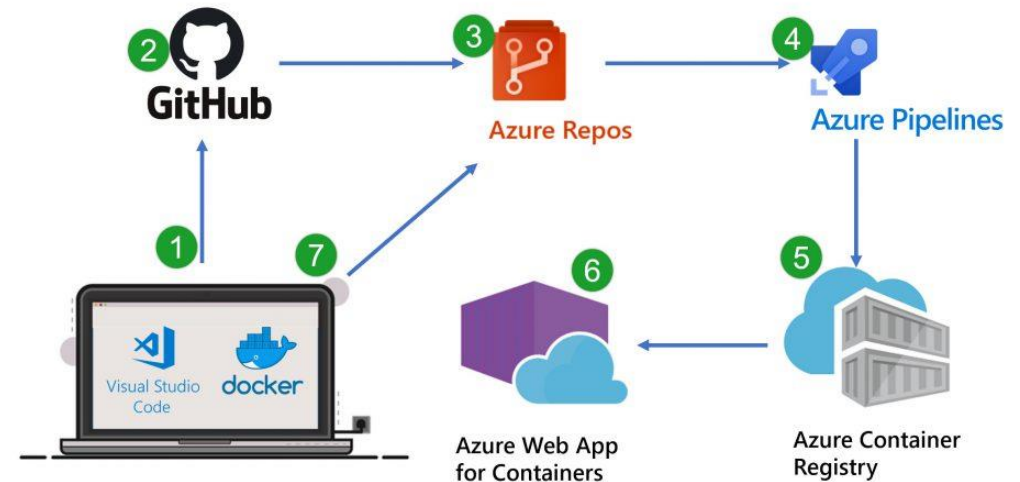
Resumen de Git flow



Azure Repos



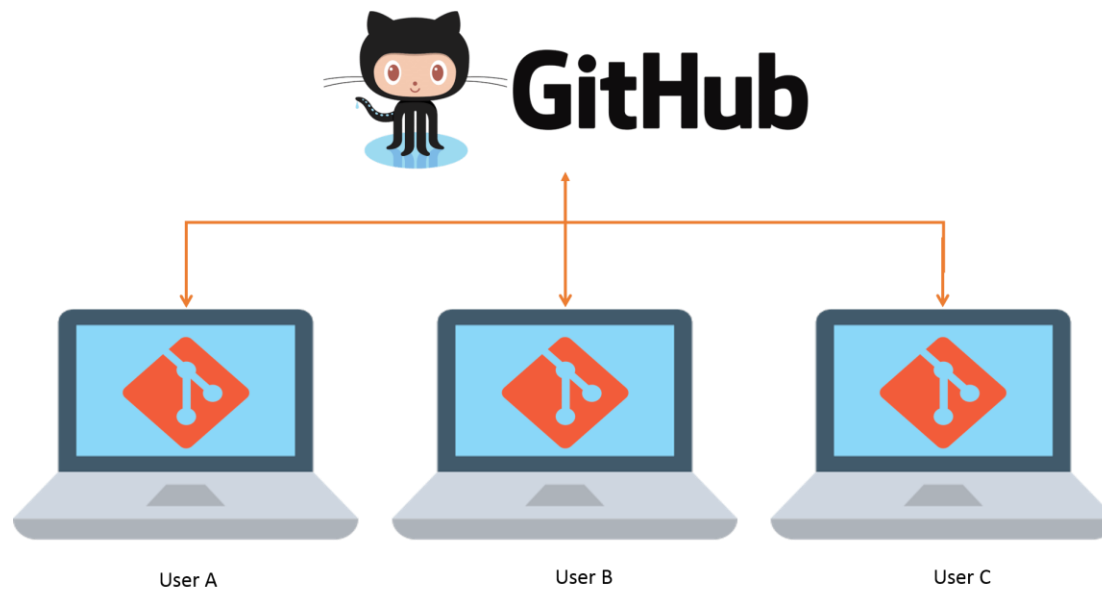
- Azure Repos es un conjunto de herramientas de control de versiones que le ayuda a administrar su código. Con Azure Repos, puede conectarse a cualquier entorno de desarrollo, revisar código con solicitudes de extracción, proteger sucursales con políticas, aislar código con bifurcaciones y hacer mucho más.





Integración con GitHub

- Azure Repos, admite Git, que es un control de versiones distribuido, y Team Foundation Version Control (TFVC), que es un control de versiones centralizado.





- Integración con Visual Studio Code, Git y Azure DevOps

The screenshot shows the Azure DevOps web interface for the 'Parts Unlimited' repository. The left sidebar contains navigation links: Overview, Boards, Repos, Files, Commits (selected), Pushes, Branches, Tags, Pull requests, Pipelines, Test Plans, and Project settings. The main area displays the 'Commits' page for the 'master' branch. A table lists the commit history:

Graph	Commit	Pull Request	Status
	Comentarios añadidos 1fd2989f Carlos Carreno Today at 11:31		✓ succeeded
	Mi commit f9c9cafe Carlos Carreno Today at 11:24		✓ succeeded
	Updated FullEnvironmentSetupMerged.json 8086ee35 Akshay H 30 de setiembre de 2019 at 09:30		
	Updated FullEnvironmentSetupMerged.param.json 58d9b878 Srivatsa Marichi 24 de noviembre de 2017 at 04:52		
	Updated FullEnvironmentSetupMerged.json bef6d46f Srivatsa Marichi 24 de noviembre de 2017 at 04:47		
	Removing ARM project 816d9487 vsts 30 de enero de 2017 at 22:47	1.0.0.25	
	Adding buildinfo target to project f731482e Colin Dembovsky 30 de enero de 2017 at 12:46		
	Fixing test framework for LUT ef0e4f4a Colin Dembovsky 27 de enero de 2017 at 01:54		
	Ready for demo		



Referencias

- <https://git-scm.com/downloads>
- <https://cleventy.com/que-es-git-flow-y-como-funciona/>

Laboratorio



- Lab Version Controlling with Git in Visual Studio Code and Azure DevOps