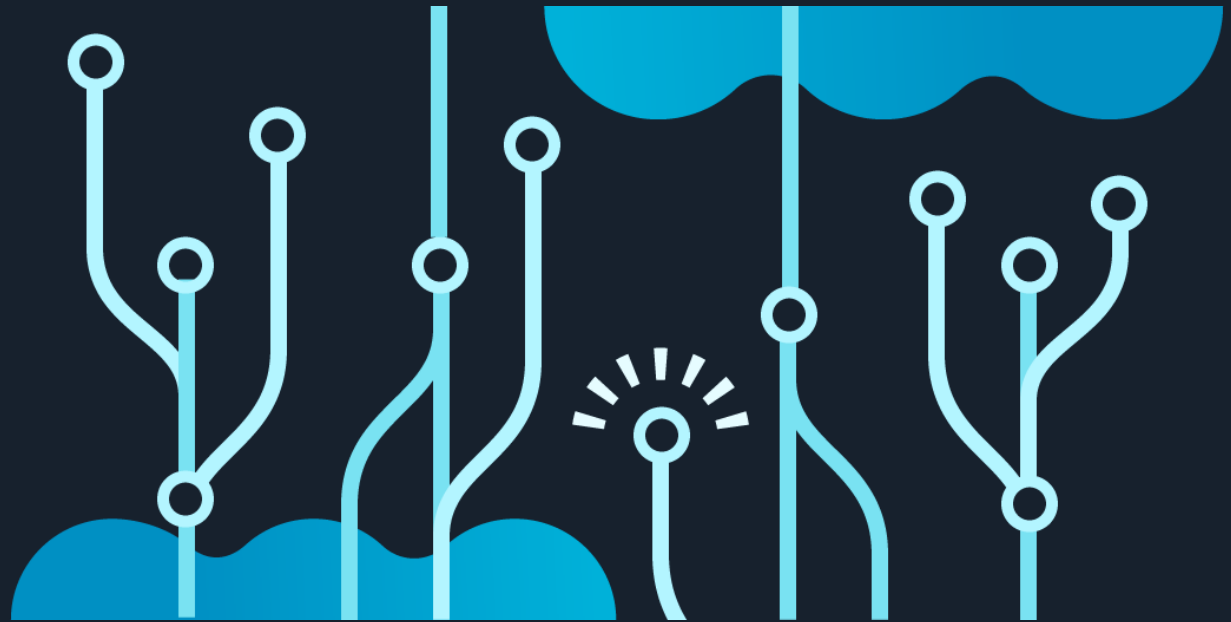


plain concepts



# GIT



# GIT

Git es una herramienta de **control de versiones** (SCM).

Sus principales características:

- **Distribuido**. Cada copia contiene todos los cambios del histórico.
- Es **liviano y rápido (para código no para binarios)**
- Desarrollo ágil con distintas **ramas de código**
- Se adapta a distintos **workflows de trabajo**
- **Open Source** con gran comunidad
- **Multipataforma**

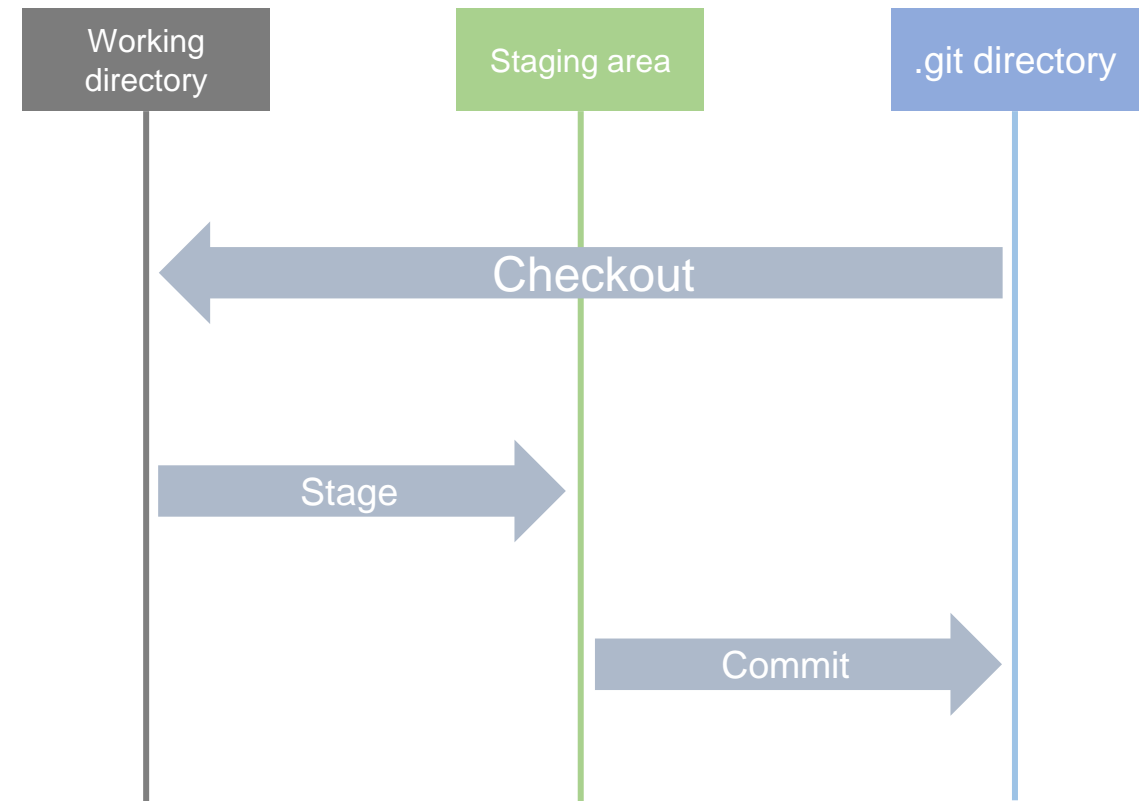
Inicialmente, Git fue creado por **Linus Torvalds** para el desarrollo del kernel de Linux.

De hecho, Git se versionó con Git! Empezando desde el 2º día de desarrollo, el código de Git está versionado.

# GIT – STAGES

Git tiene tres etapas donde se aplican cambios:

- Working directory: Cambios actuales en desarrollo
- Staging area: Cambios que queremos trackear al histórico de git
- .git directory: Repositorio donde se guarda el histórico de cambios

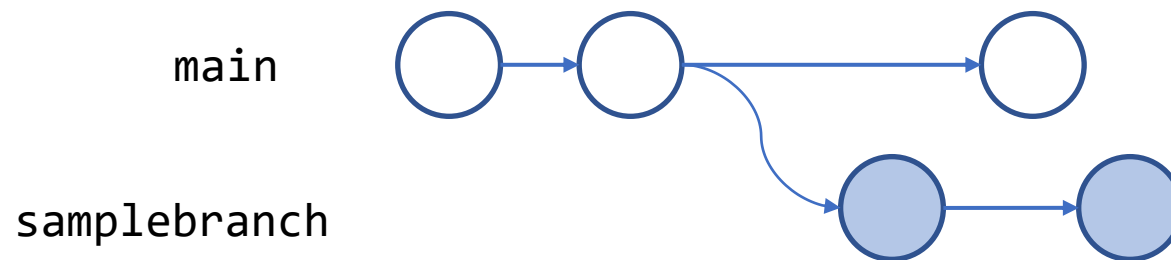


# GIT – RAMAS (BRANCHES)

Se pueden crear ramas a partir de la versión principal añadiendo cambios de contenido. Las ramas se pueden integrar unas con otras siempre que no hayan conflictos.

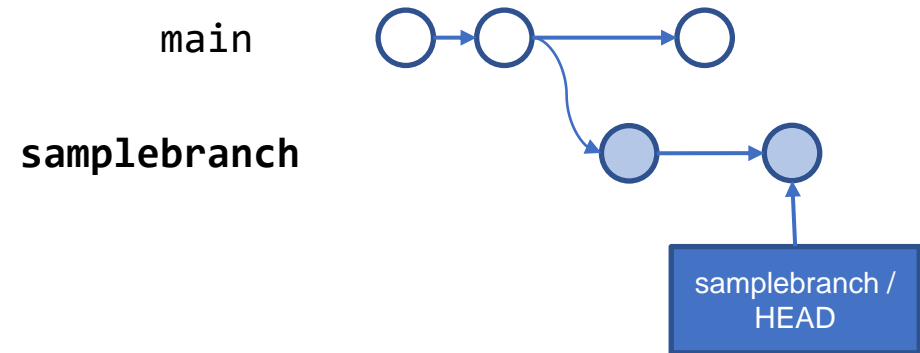
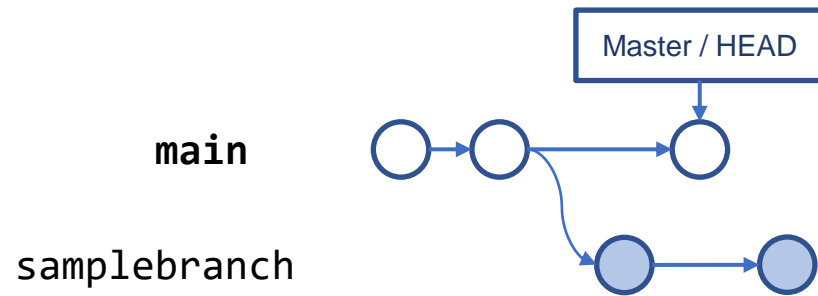
En caso de conflictos, es necesario hacer un Merge para decidir con qué versión de cambios nos queremos quedar.

En git, la convención y la rama por defecto es la rama **main** (aunque podemos cambiar la rama por defecto si así lo deseamos).



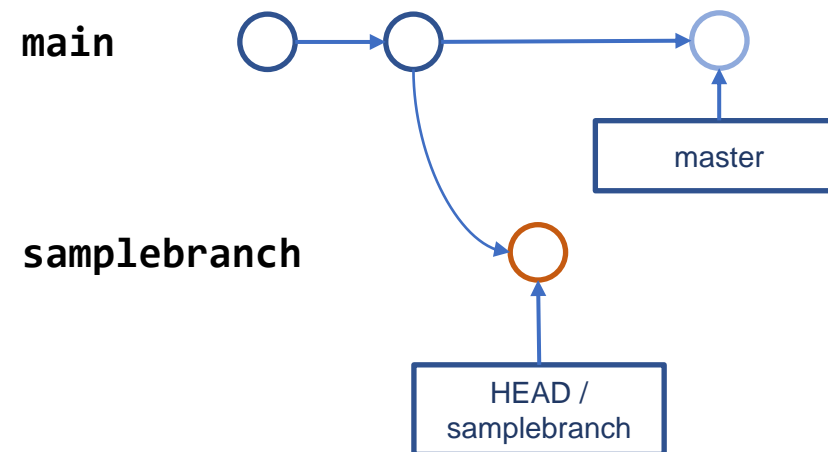
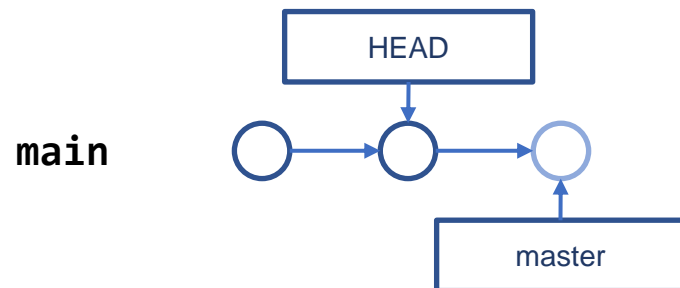
# GIT – HEAD

- El HEAD es un **puntero al último commit** donde se encuentra la copia local.
- Cuando cambiamos de rama, el HEAD apunta al último commit de esta



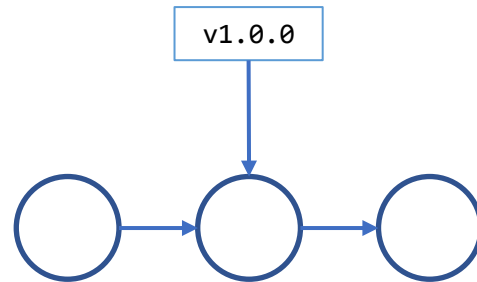
# GIT – HEAD

- Si el HEAD no apunta a una referencia de una rama, estamos en estado **Detached Head**.
  - Esto puede traer problemas si se introducen nuevos cambios ya que al cambiar de rama, no habrá **ninguna rama que haga referencia a estos nuevos commits** y se perderán.
  - Por eso, para introducir nuevos cambios a partir de un commit anterior, será **necesario crear una rama** después de mover el HEAD al commit deseado



# GIT – TAGS

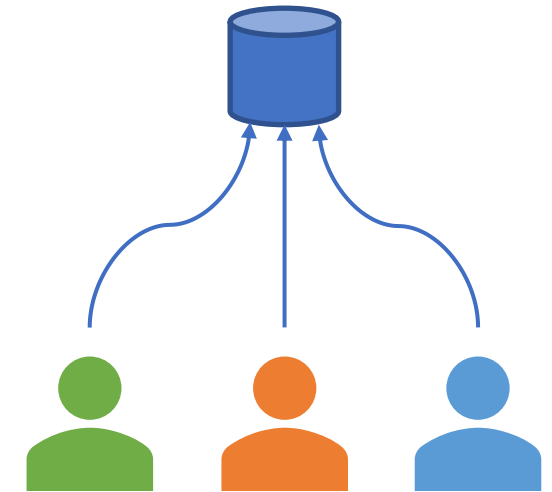
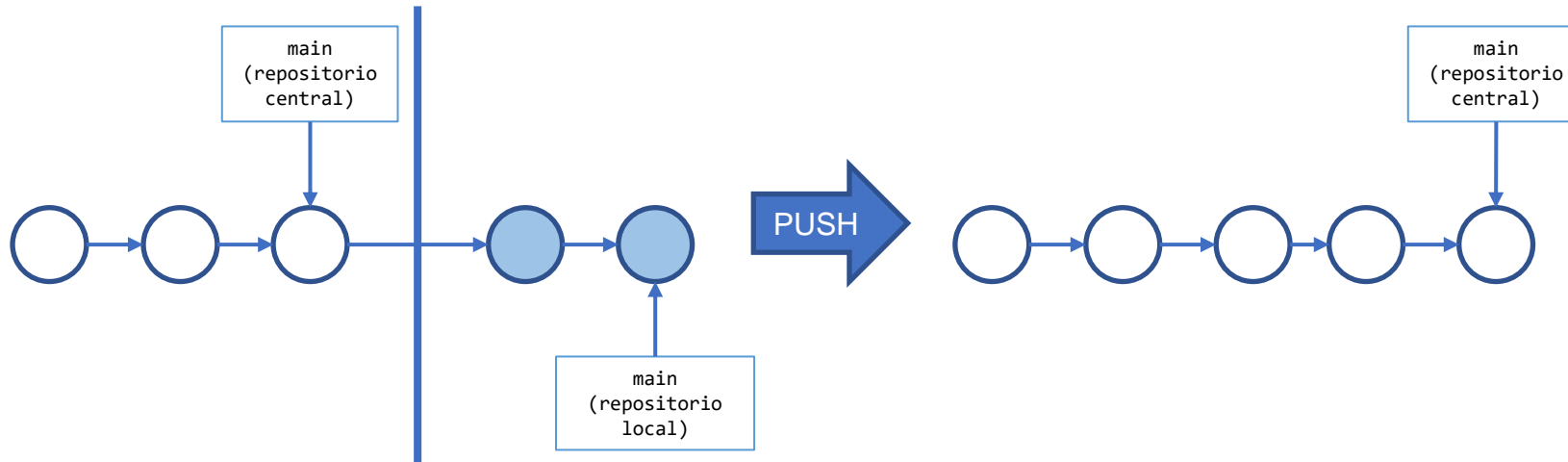
- Marcar versiones concretas de código de forma que sea fácil identificarlas.
- Útil para versionado de releases





# GIT - FLUJOS

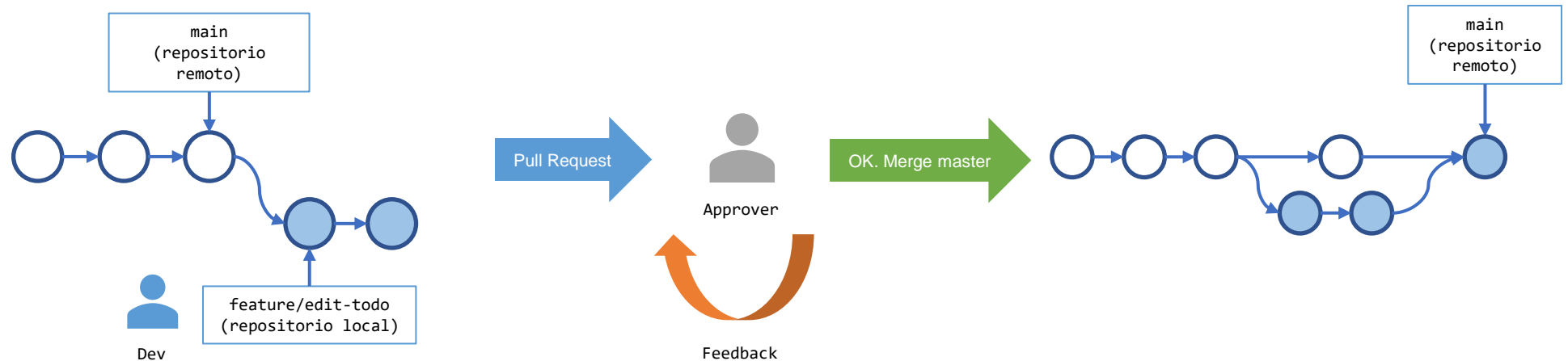
- Centralized Workflow
  - Los cambios se aplican integrándose directamente contra un único repositorio remoto en main
  - Sólo existe **1 rama**
  - Parecido a subversion, TFS VC...
  - **Difícil de trabajar en equipo**



# GIT - FLUJOS

- Feature Branch Workflow

- El desarrollador crea una rama local a partir de main donde se desarrollará una **nueva funcionalidad**.
- Cuando el desarrollo esté listo, la **rama se integrará a main** a través de un flujo: **Pull Requests**
- Común en equipos estables, proyectos de código cerrado.
- Promueve la **colaboración con miembros del equipo**
- Azure DevOps, BitBucket, etc.



# GIT - FLUJOS

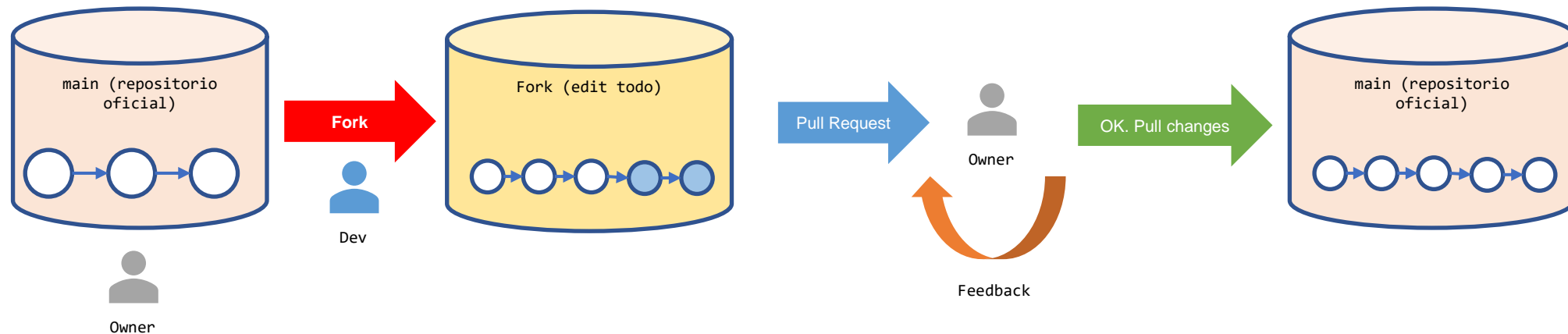
- Feature Branch Workflow
  - La versión en master debe ser **candidata** para una nueva **release**
  - Una **feature branch** contiene **una** nueva **funcionalidad** para la aplicación.
  - Al integrar la feature, **la rama debe eliminarse**
  - Ciclo de vida **corto**
  - Ramas **integradas** con la versión en **main**
  - En caso de **bugs en main**, hacer una **rama de hotfix** desde el commit de la última release y desplegar nueva versión con el parche.

```
main  
=> feature/edit-todos  
=> hotfix/delete-todo  
=> hotfix/v1.2  
=> hotfix/sprint-34
```

# GIT - FLUJOS

- Forking Workflow

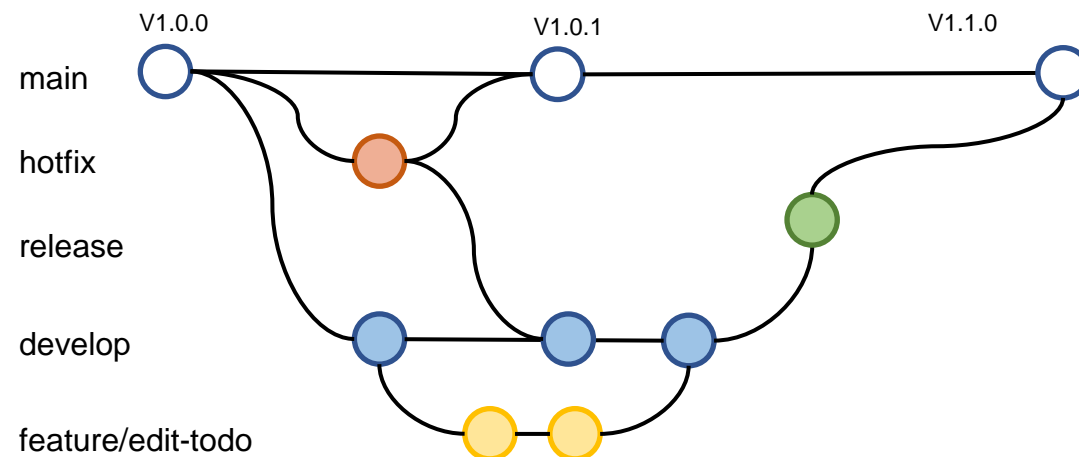
- Un desarrollador realiza un fork (copia) del repositorio original para hacer cambios. Siguiendo un flujo de petición, se pide al owner del repositorio original que haga pull del fork para integrar los nuevos cambios.
- Común en proyectos Open Source y muy extendido gracias a GitHub con sus Pull Request



# GIT - FLUJOS

- Gitflow Workflow

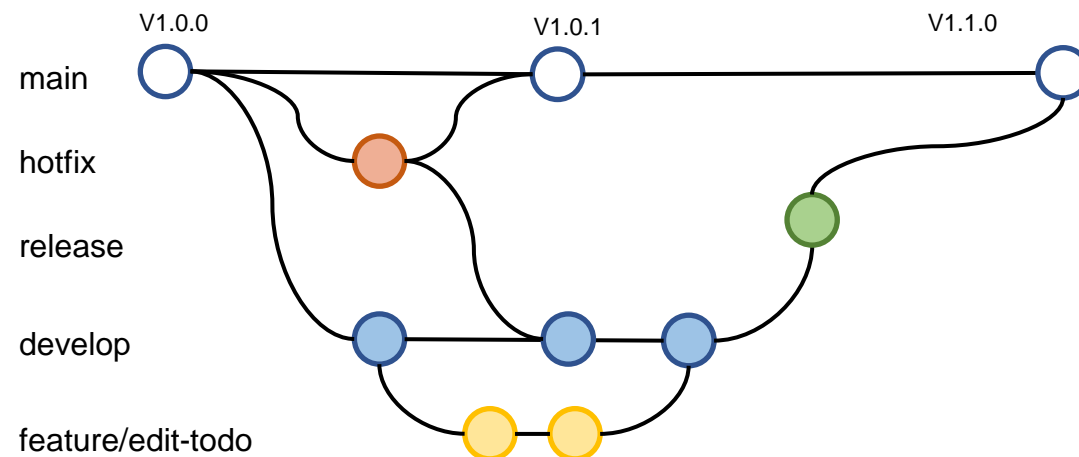
- **Main Branch:** Histórico oficial de releases del proyecto. Ideal marcar con tags las versiones.
- **Develop Branch:** Aquí se integran nuevos desarrollos
- **Feature branches:** Nueva funcionalidad a partir de develop
- **Hotfix branches:** Cuando hay un bug de una versión publicada, se hace un hotfix desde main y integra en main y develop.
- **Release branches:** Cuando develop tiene funcionalidades para ser desplegadas, se hace una rama release candidata para publicarse y finalmente se integrará a main como una nueva versión.



# GIT - FLUJOS

- Gitflow Workflow

- **Main Branch:** Histórico oficial de releases del proyecto. Ideal marcar con tags las versiones.
- **Develop Branch:** Aquí se integran nuevos desarrollos
- **Feature branches:** Nueva funcionalidad a partir de develop
- **Hotfix branches:** Cuando hay un bug de una versión publicada, se hace un hotfix desde main y integra en main y develop.
- **Release branches:** Cuando develop tiene funcionalidades para ser desplegadas, se hace una rama release candidata para publicarse y finalmente se integrará a main como una nueva versión.



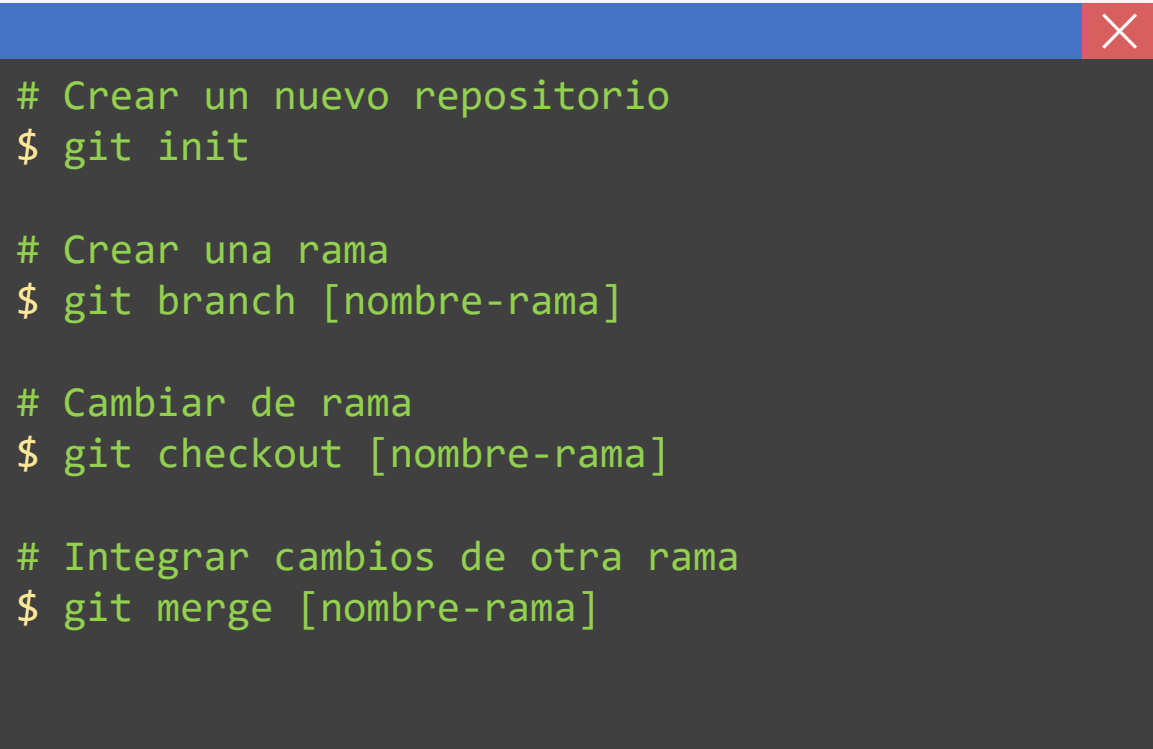
# DIFERENCIAS CON TFS



- Distribuído
- Branch friendly
- Funciona offline
- Flujos de trabajo eficientes
- Rápido
- Multiplataforma
- Gran comunidad
- Menos conflictos con merge
- Fácil cambiar de tarea
- Apuesta actual de Microsoft

- Centralizado
- Sistema de ramas poco flexible
- Sólo funciona online
- Poca flexibilidad para hacer flujos de trabajo
- Lento
- Windows only
- Poca comunidad
- Más conflictos con merge
- Difícil cambiar de tarea
- Futuro incierto

# GIT EN LA CONSOLA – GESTIÓN DEL REPO



```
# Crear un nuevo repositorio
$ git init

# Crear una rama
$ git branch [nombre-rama]

# Cambiar de rama
$ git checkout [nombre-rama]

# Integrar cambios de otra rama
$ git merge [nombre-rama]
```



# GIT EN LA CONSOLA – GESTIÓN DE CAMBIOS

```
# Añadir a stage un fichero creado/modificado
$ git add nombre-fichero.txt

# Añadir a stage todos los ficheros con cambios
$ git add *

# Quitar de stage todos los ficheros cambios
$ git rm --cached *

# Trackear cambio (COMMIT)
$ git commit -m "Añadido fichero"

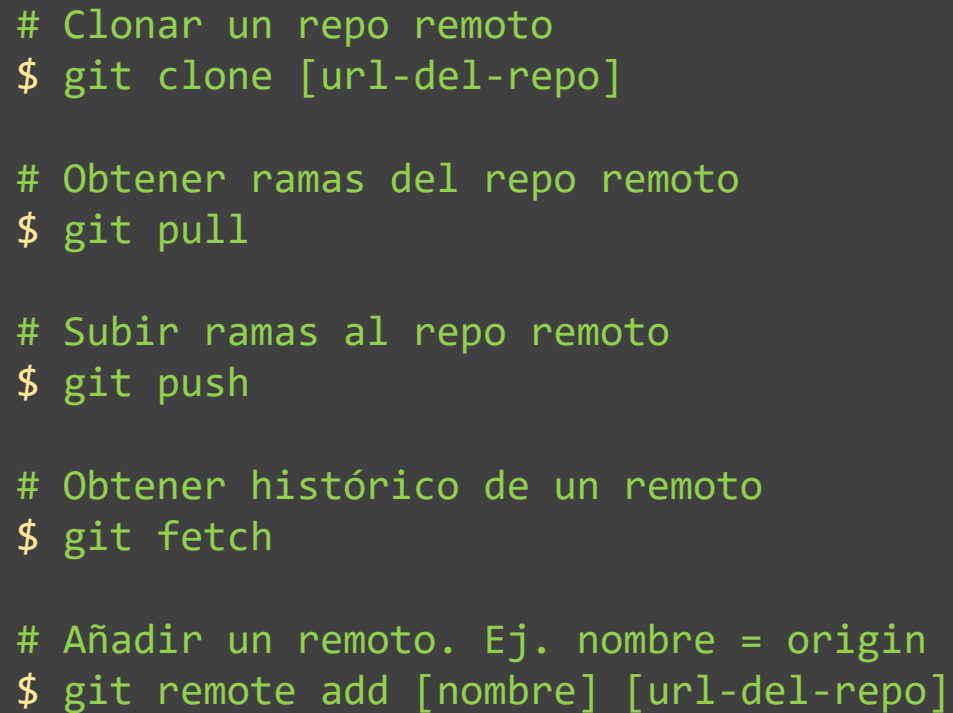
# Descartar todos los cambios no commiteados
$ git reset --hard
```

```
# Listar cambios actuales
$ git status

# Listar diferencias
$ git diff

# Listar histórico
$ git log
```

# GIT EN LA CONSOLA – TRABAJANDO CON REMOTOS



```
# Clonar un repo remoto
$ git clone [url-del-repo]

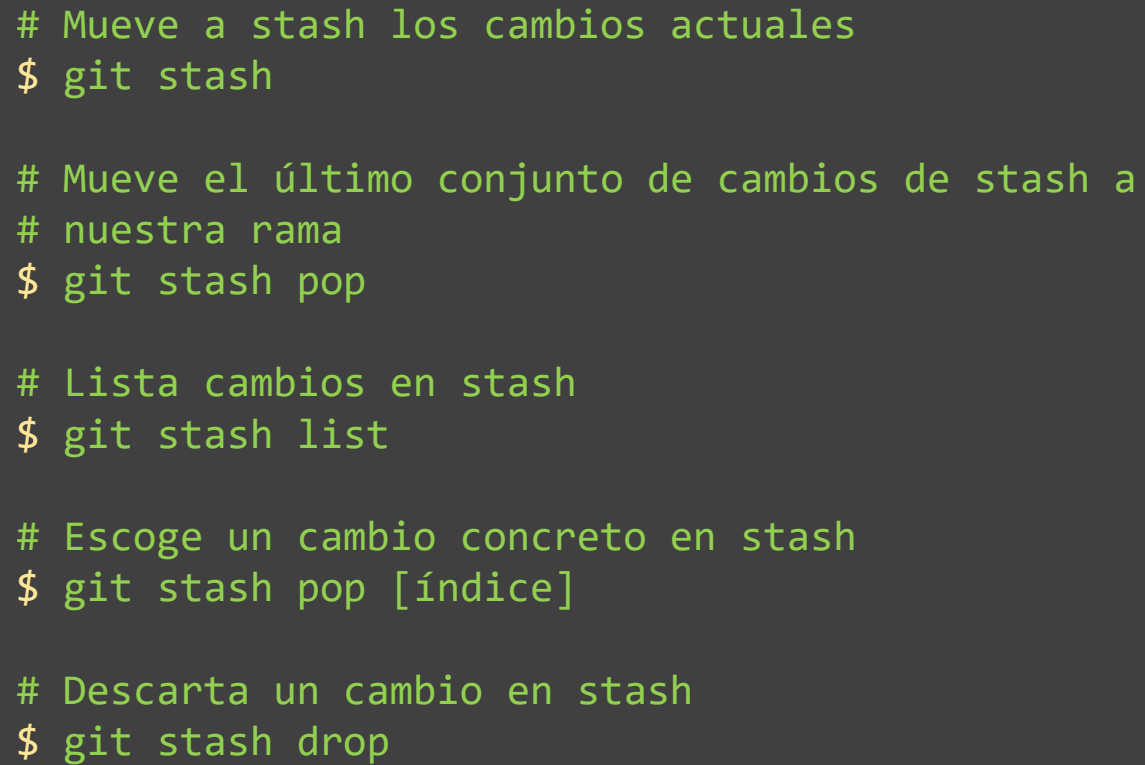
# Obtener ramas del repo remoto
$ git pull

# Subir ramas al repo remoto
$ git push

# Obtener histórico de un remoto
$ git fetch

# Añadir un remoto. Ej. nombre = origin
$ git remote add [nombre] [url-del-repo]
```

# GIT EN LA CONSOLA – STASH



```
# Mueve a stash los cambios actuales
$ git stash

# Mueve el último conjunto de cambios de stash a
# nuestra rama
$ git stash pop

# Lista cambios en stash
$ git stash list

# Escoge un cambio concreto en stash
$ git stash pop [índice]

# Descarta un cambio en stash
$ git stash drop
```

# GIT - CONFIGURACIÓN

- .gitignore
  - Configuración de ficheros a ignorar en el repositorio
- .gitattributes
  - Configuración git del repositorio (normalización de espacios, contenido ficheros, etc.)
- Git user config
  - Configuración de metadatos del usuario que se muestran en el histórico (Nombre, email, etc.)
  - Ej.: `git config user.name "John Smith"`
  - Ej.: `git config user.email "abc@xyz.com"`

## .gitignore

```
**/bin
**/*.zip
secretos.txt
output/*.min.css
```

## .gitattributes

```
* text=auto
*.cs text
*.sln text eol=crlf
*.jpg binary
```

```
# Crear un nuevo repositorio
$ git init

# Crear una rama
$ git branch [nombre-rama]

# Cambiar de rama
$ git checkout [nombre-rama]

# Integrar cambios de otra rama
$ git merge [nombre-rama]

# Clonar un repo remoto
$ git clone [url-del-repo]

# Obtener ramas del repo remoto
$ git pull

# Subir ramas al repo remoto
$ git push

# Obtener histórico de un remoto
$ git fetch

# Añadir un remoto. Ej. nombre = origin
$ git remote add [nombre] [url-del-repo]
```

```
# Añadir a stage un fichero creado/modificado
$ git add nombre-fichero.txt

# Añadir a stage todos los cambios
$ git add *

# Quitar de stage todos los cambios
$ git rm --cached *

# Trackear cambio (COMMIT)
$ git commit -m "Añadido fichero"

# Descartar todos los cambios no commiteados
$ git reset --hard

# Listar cambios actuales
$ git status

# Listar diferencias
$ git diff

# Listar histórico
$ git log
```

```
# Mueve a stash los cambios actuales
$ git stash

# Mueve los cambios de el último cambio
# de stash a nuestra rama
$ git stash pop

# Lista cambios en stash
$ git stash list

# Escoge un cambio concreto en stash
$ git stash pop [índice]

# Descarta un cambio en stash
$ git stash drop
```

**.gitignore**

```
*/bin
*/*.zip
secretos.txt
output/*.min.css
```

# GIT EN LA CONSOLA

Algunas chuletas más:

- <https://education.github.com/git-cheat-sheet-education.pdf>
- <https://the-awesome-git-cheat-sheet.com/>

# ¿CONSOLA ? YO PREFIERO ALTERNATIVAS

Existen muchos clientes de git:

- Visual Studio (hace de todo)
- Sourcetree
- Gitkraken
- TortoiseGit
- SmartGit
- ...

# JUGANDO CON GIT



# CASOS REALES (NOS HA PASADO A TODOS 😊)

- He hecho commit en otra rama!
- Como revierto un cambio?
- Como puedo volver a una versión anterior?



Azure Boards



Azure Pipelines



Azure Repos



Azure Test Plans



Azure Artifacts



# Azure DevOps

# AZURE DEVOPS - SERVICIOS



Azure Boards



Azure Pipelines



Azure Repos

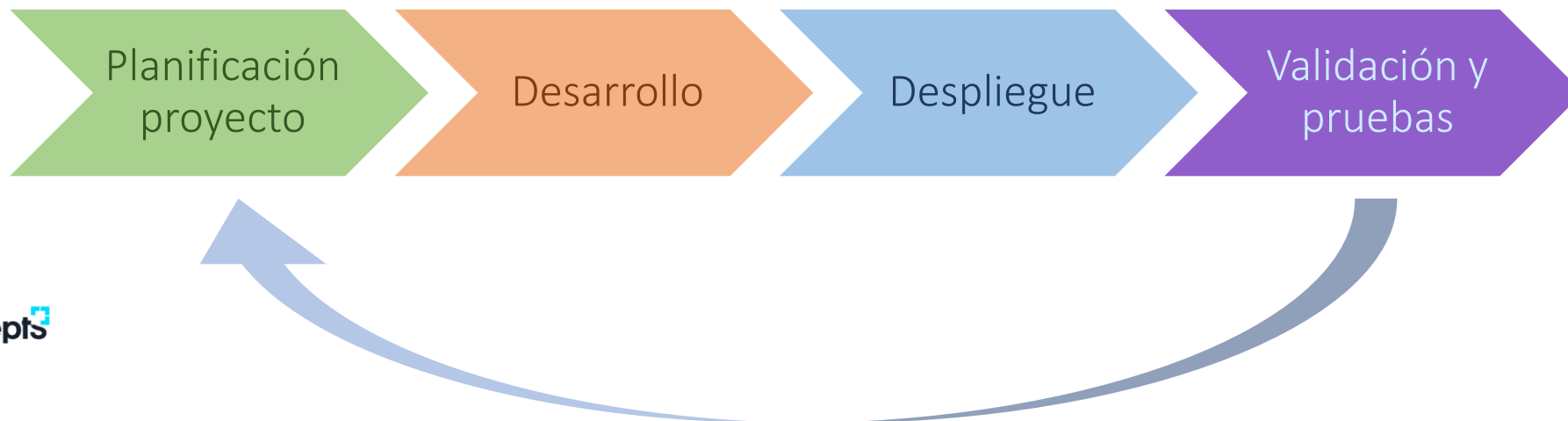


Azure Test Plans



Azure Artifacts

Solución que abarca las etapas de un proyecto de forma ágil.



REPOS



Azure DevOps



Azure Repos

# AZURE DEVOPS - REPOS



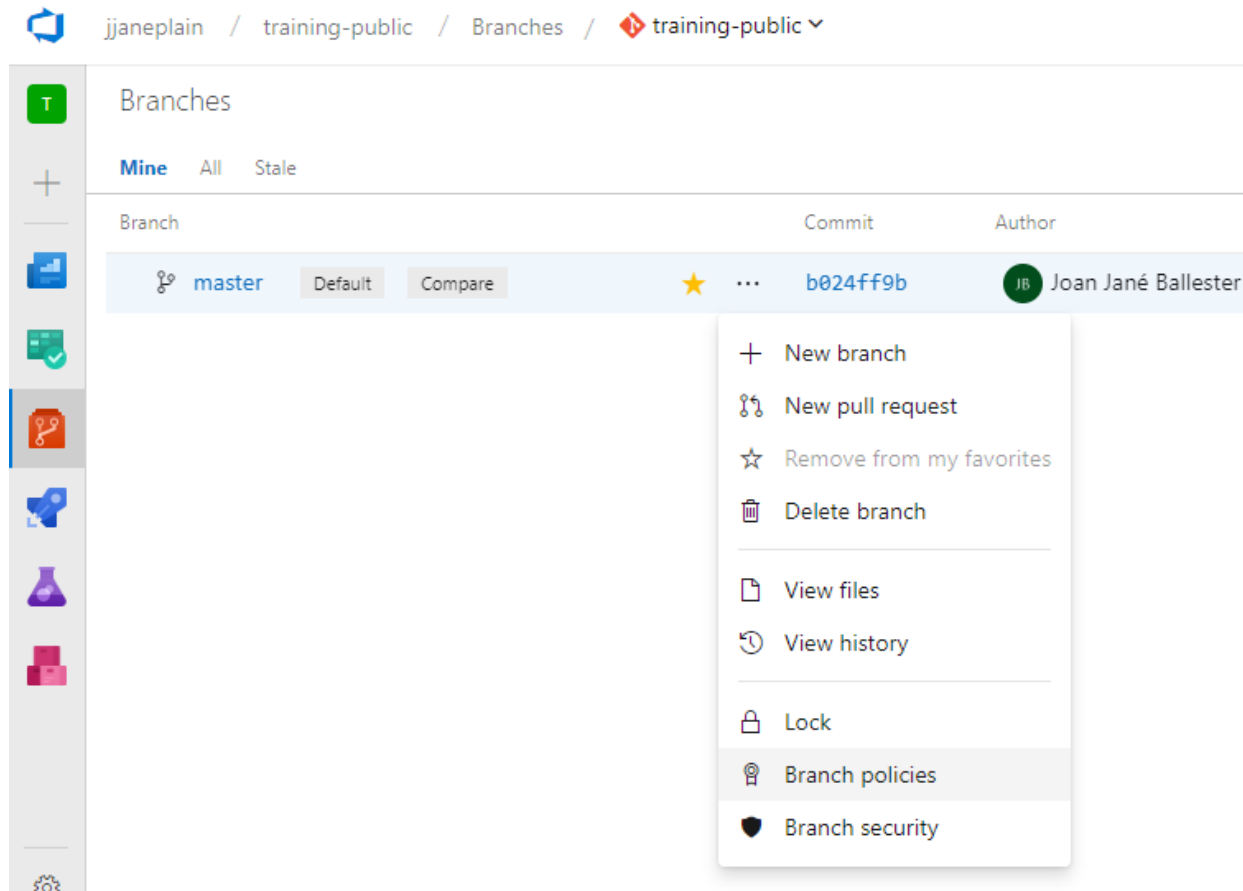
Azure Repos

- Opción para hostear múltiples repos git
- Políticas de código por rama
- Políticas de seguridad por equipos
- Trabajo colaborativo con Pull Requests
- Vinculado con el backlog, builds y releases

# AZURE DEVOPS – PULL REQUESTS

- La rama en cuestión (master) se protege con Pull Requests y no se permite modificaciones directas
- El equipo revisa los cambios de una rama antes de integrarlos en master
- Se pueden escribir comentarios en el código hacer tracking del estado
- Se puede vincular con una build para asegurar que compila, pasa los test, etc
- Se puede vincular con work items del backlog
- Existen políticas para establecer unas prácticas dentro de un equipo

# AZURE DEVOPS – PULL REQUESTS – BRANCH POLICIES



# AZURE DEVOPS – PULL REQUESTS – BRANCH POLICIES

- ☒ **Require a minimum number of reviewers**  
Require approval from a specified number of reviewers on pull requests.

Minimum number of reviewers

- ☒ Allow users to approve their own changes.
- ☐ Allow completion even if some reviewers vote "Waiting" or "Reject".
- ☐ Reset code reviewer votes when there are new changes.

- ☒ **Check for linked work items**  
Encourage traceability by checking for linked work items on pull requests.

Policy requirement

- ☐ Required  
Block pull requests from being completed unless they have at least one linked work item.
- ☒ Optional  
Warn if there are no linked work items, but allow pull requests to be completed.

- ☒ **Check for comment resolution**  
Check to see that all comments have been resolved on pull requests.

Policy requirement

- ☒ Required  
Block pull requests from being completed while any comments are active.
- ☐ Optional  
Warn if any comments are active, but allow pull requests to be completed.

- ☐ **Enforce a merge strategy**  
Require a specific type of merge when pull requests are completed.




# AZURE DEVOPS – PULL REQUESTS – BRANCH POLICIES

## Build validation

Validate code by pre-merging and building pull request changes

+ Add build policy

Build pipeline	Requirem...	Path filter	Expiration	Trigger	
 training-public-CI	Required	No filter	Expires after 12 hours	Automatic	<input checked="" type="checkbox"/> Enabled

## Automatically include code reviewers

Include specific users or groups in the code review based on which files changed.

+ Add automatic reviewers

Reviewer(s)	Requirem...	Path filter	
 [training-public]\training	Required	No filter	<input checked="" type="checkbox"/> Enabled

BUILDS AND  
RELEASES  
CI / CD



Azure DevOps



Azure Pipelines

# AZURE DEVOPS – CI/CD



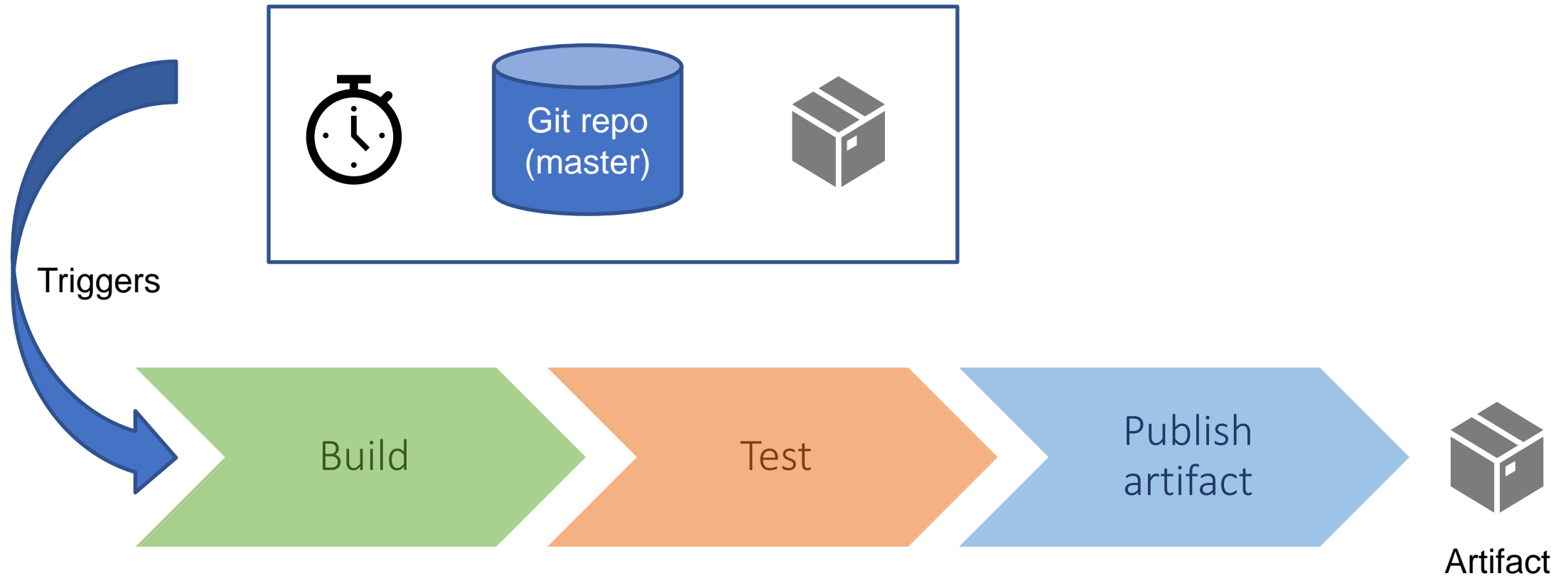
## Continuous Integration

*“Práctica de desarrollo que busca que **los desarrolladores integren su código** en un repositorio compartido **cuánto antes** para **detectar errores** antes y más fácilmente”*

## Continuous Delivery

*“Práctica en la entrega del software que busca **hacer ciclos cortos de entregas** y que permita desplegar en cualquier momento”*

# AZURE DEVOPS – BUILDS



# AZURE DEVOPS – BUILDS – TASKS

- Basadas en tareas ejecutadas en uno o más agentes
- El orden de las tareas es relevante
- Sistema condicional: Controlar si una tarea se ejecuta o no
  - Por defecto, si una tarea falla, las otras se ignoran y la build falla
  - Condiciones custom: Ej.: Motivo de la build es una PR
- Existe un Marketplace con un amplio catalogo de tareas pero además de esto se pueden crear y publicar tareas personalizadas
- Hay tareas genéricas de ejecución de scripts

plain concepts



Customizable. Cross platform.

# AZURE DEVOPS – BUILDS - TASKS

training-public-CI

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ...

Pipeline  
Build pipeline

Get sources  
training-public master

Agent job  
Run on agent

- Restore nuget packages  
.NET Core
- Build  
.NET Core
- Test  
.NET Core
- Publish web project  
.NET Core
- Publish Artifact  
Publish Build Artifacts**

Publish Build Artifacts ⓘ

Task version 1.\* [Link settings](#) [View YAML](#)

Display name \*  
Publish Artifact

Path to publish \* ⓘ  
\$(Build.ArtifactStagingDirectory) ...

Artifact name \* ⓘ  
drop

Artifact publish location \* ⓘ  
Azure Pipelines/TFS

Control Options ▾

Output Variables ▾

# AZURE DEVOPS – BUILDS – TRIGGERS

- Continuous Integration
  - Cuando hay un cambio en una rama concreta del repositorio
- Scheduled
  - Cada cierto tiempo. Ej. Cada noche
- Build completion
  - Cuando otra build ha terminado

# AZURE DEVOPS – BUILDS - VARIABLES

- Las variables sirven para parametrizar los valores de las tareas
- Posibilidad de que se especifique en tiempo de encolar la build
- Pueden ser secretas

🏠 ... > training-public-CI

Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ...

Pipeline variables	Name ↑	Value
Variable groups	BuildConfiguration	release
Predefined variables <a href="#">🔗</a>	system.collectionId	609dca5f-f055-4080-a386-eedab37f4280
	system.debug	false

Arguments ⓘ

```
--configuration $(BuildConfiguration) --output $(build.artifactstagingdirectory)|
```



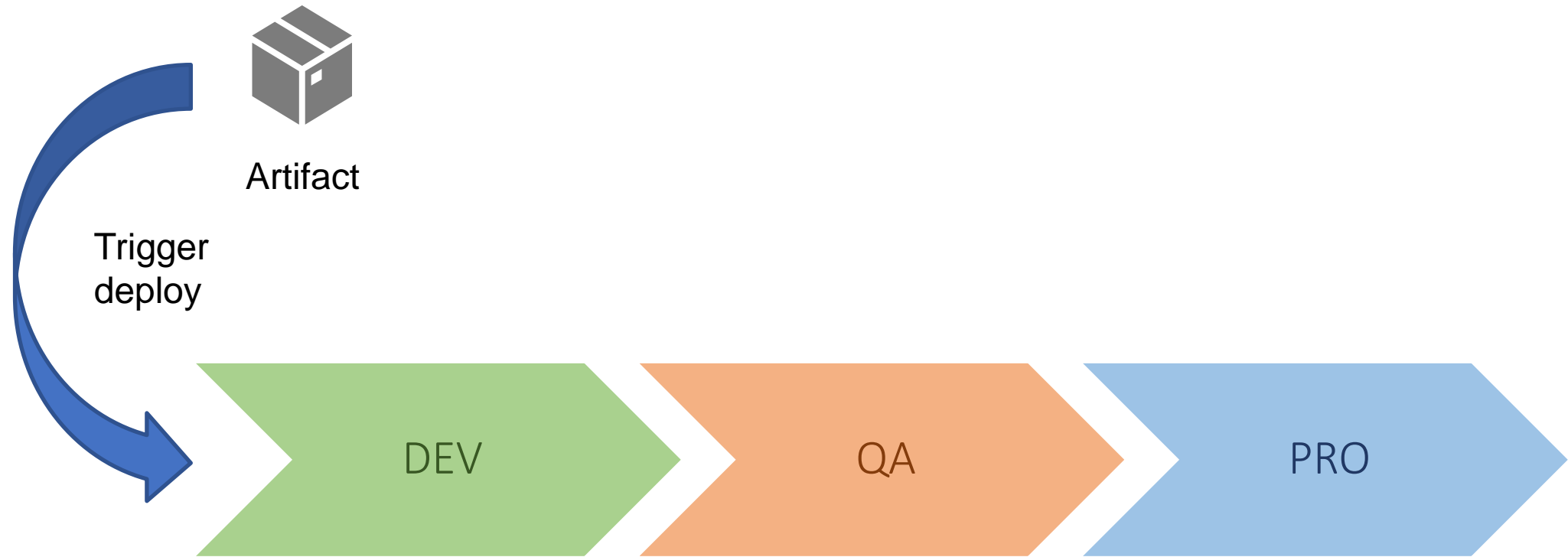
# AZURE DEVOPS – BUILDS

- Histórico de cambios de la build
- Política de retenciones de artefactos

# AZURE DEVOPS – BUILDS – STAGES/JOB/STEPS

- Stages: se pueden organizar las compilaciones por etapas o fases, estas etapas son las divisiones principales de una pipeline, como por ejemplo: "compilar la aplicación", "ejecutar las pruebas" y "desplegar en desarrollo". Cada canalización tiene al menos una fase aunque no la defina explícitamente.
- Jobs: se pueden organizar las pipelines en trabajos. Un trabajo es una serie de pasos que se ejecutan secuencialmente como una unidad.
- Steps: son la unidad mas pequeña, básicamente podríamos decir que son cada una de las actividades que se realizan.

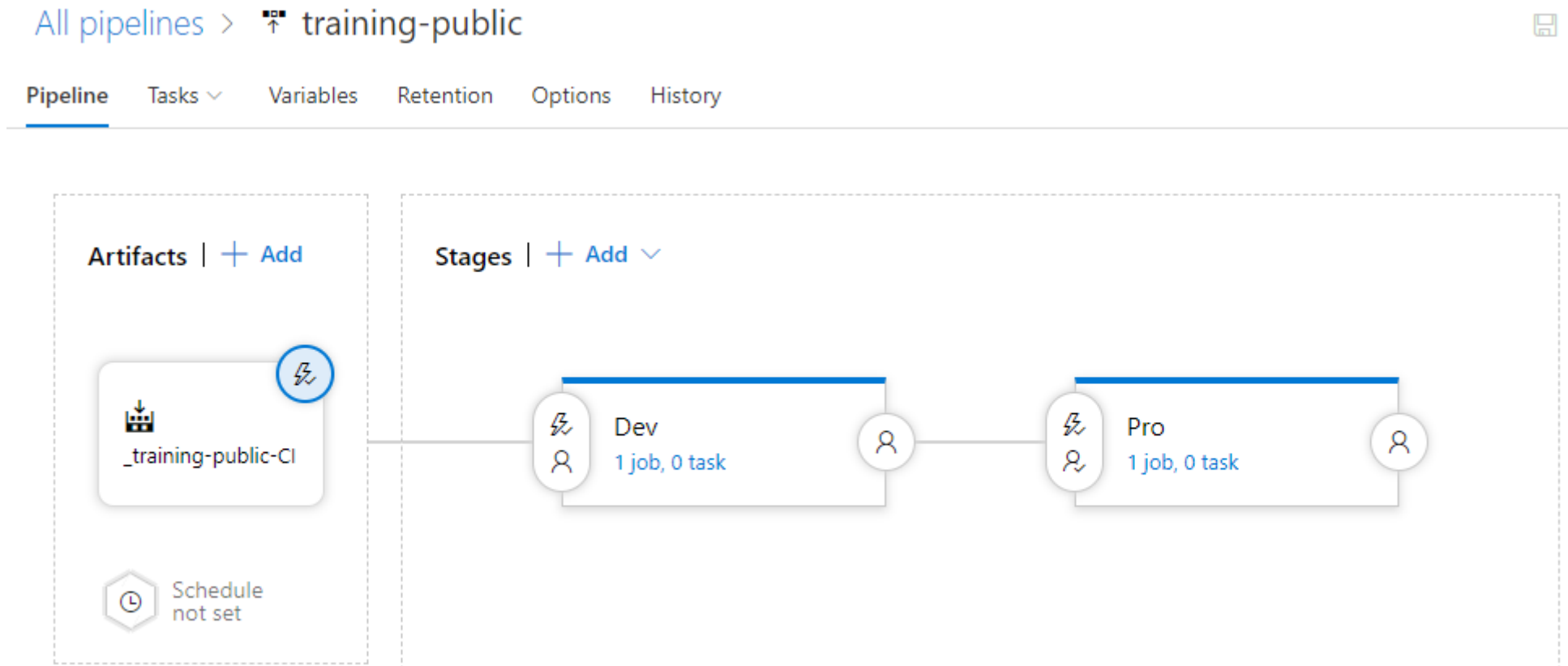
# AZURE DEVOPS – RELEASES



# AZURE DEVOPS – RELEASES

- Mismo motor de tareas que las Builds
- El enfoque ideal debería ser 1 release, múltiples entornos (stages)
- Workflow de aprobación entre stages
- El trigger es un artefacto, el artefacto es el binario de la aplicación que se desplegará.
- Triggers entre stages
  - Después de una stage
  - Manual
  - Al crear la release

# AZURE DEVOPS – RELEASES - STAGES



# AZURE DEVOPS – RELEASES – STAGE TRIGGER

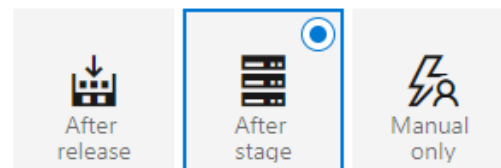
## Pre-deployment conditions

Pro

## Triggers ^

Define the trigger that will start deployment to this stage

Select trigger ⓘ



Stages ⓘ



☐ Trigger even when the selected stages partially succeed ⓘ

Artifact filters ⓘ

☐ Disabled

Schedule ⓘ




☐ Disabled

Pull request deployment ⓘ


☐ Disabled



ⓘ This release definition is currently not configured to trigger pull request based releases. To configure, toggle on a pull request trigger in at least one of the added artifacts.


# AZURE DEVOPS – RELEASES - APPROVALS

 Pre-deployment approvals   Enabled



Select the users who can approve or reject deployments to this stage

Approvers 

 [training-public]\training-public Team  Search users and groups for approve

Timeout 

Approval policies

- ☐ The user requesting a release or deployment should not approve it
- ☐ Revalidate identity of approver before completing the approval. 
- ☐ Skip approval if the same approver approved the previous stage 

# AZURE DEVOPS – RELEASES – VARIABLES POR STAGE

Variables Retention Options History

Filter by keywords Scope X

Name	Value	Lock	Scope	Settable at release time
DbConnectionString	La connection string de dev		Dev	▼
DbConnectionString	*****		Pro	▼ <input type="checkbox"/>

+ Add








# AZURE DEVOPS – RELEASES – TASKS POR STAGE

Pipeline **Tasks** ▾ Variables Retention Options History

**dev** ...  
Deployment process

**Agent phase** +  
Run on agent

-  **arm**  
Azure Resource Group Deployment
-  **deploy app**  
Azure App Service Deploy
-  **Smoke Web Test**  
Smoke Web Test
-  **swap**  
Azure App Service Manage
-  **Update ApiManagement**  
Azure PowerShell

# AZURE DEVOPS – OTROS SERVICIOS



## Azure Test Plans

- Gestión de pruebas
- Automatizadas
- Manuales
- Tracking



## Azure Artifacts

- Solución para feeds de paquetes
  - Nuget
  - Npm
  - Maven



1 In any browser navigate to homepage and log in



2 Navigate to a product and add it to cart



3 Check if cart is populated with Item



This results in a 404 once you try to navigate to the shopping cart



FabrikamVersions.Widgets

Version 4.0.0-ci-20161228-215550



msft-colors

Version 1.0.0



my-first-unpack

Version 1.0.0



MyGroup:myFirstApp

Version 1.0-SNAPSHOT

¿PREGUNTAS?

GRACIAS POR VUESTRO  
TIEMPO!