Trabajando con Ramas

Branches, Git Flow y Estrategias de Desarrollo

Clase 4 · Trabajando con Ramas

GLUD — Grupo GNU/Linux Universidad Distrital

Control de Versiones y Desarrollo Colaborativo

Mapa de ruta

1. ¿Qué son las Ramas?
2. Comandos Básicos
3. Estrategias de Ramas
4. Git Flow
5. Feature Branch Workflow
6. Comparación y Buenas Prácticas

1

¿Qué son las Ramas?

El Concepto de Ramas

Definición

Una **rama** (branch) es una línea independiente de desarrollo que permite trabajar en características sin afectar el código principal.

El Concepto de Ramas

Definición

Una **rama** (branch) es una línea independiente de desarrollo que permite trabajar en características sin afectar el código principal.

¿Por qué usar ramas?

- Aislar nuevas características
- Experimentar sin riesgos
- Trabajo paralelo en equipo
- Mantener código estable

El Concepto de Ramas

Definición

Una **rama** (branch) es una línea independiente de desarrollo que permite trabajar en características sin afectar el código principal.

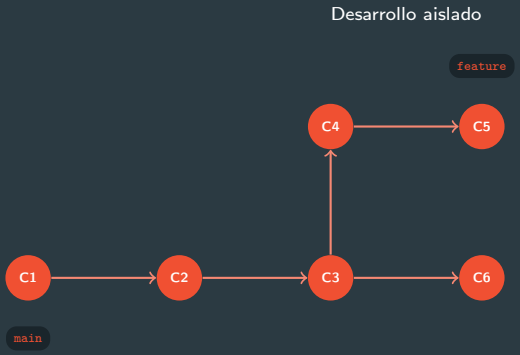
¿Por qué usar ramas?

- Aislar nuevas características
- Experimentar sin riesgos
- Trabajo paralelo en equipo
- Mantener código estable

Escenarios comunes:

- Desarrollar nueva funcionalidad
- Corregir bugs urgentes
- Probar ideas experimentales
- Preparar releases

Visualizando Ramas



Las ramas permiten desarrollo paralelo sin interferencias

> *A branch in Git is simply a lightweight movable pointer to a commit.*
>
> — Pro Git Book

2

Comandos Básicos

Ver Ramas Existentes

git branch

Lista todas las ramas locales. La rama actual se marca con *

Ver Ramas Existentes

git branch

Lista todas las ramas locales. La rama actual se marca con *

Listar ramas locales

```
git branch
```

Ver Ramas Existentes

git branch

Lista todas las ramas locales. La rama actual se marca con *

Listar ramas locales

```
git branch
```

Salida

```
  develop
* main
feature/login
```

Ver Ramas con Detalles

Ver todas las ramas

```
git branch --all
```

Ver con último commit

```
git branch -v
```

Crear Ramas

1. Crear rama sin cambiar a ella

```
git branch nombre-rama
```

Crear Ramas

1. Crear rama sin cambiar a ella

```
git branch nombre-rama
```

2. Crear y cambiar a la rama (método clásico)

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

Crear Ramas

1. Crear rama sin cambiar a ella

```
git branch nombre-rama
```

2. Crear y cambiar a la rama (método clásico)

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

3. Crear y cambiar a la rama (método moderno)

```
git switch -c nombre-rama
```

Cambiar entre Ramas

Método clásico

```
git checkout nombre-rama
```

Comando tradicional

Cambiar entre Ramas

Método clásico

```
git checkout nombre-rama
```

Comando tradicional

Método moderno

```
git switch nombre-rama
```

Más intuitivo y seguro

Cambiar entre Ramas

Método clásico

```
git checkout nombre-rama
```

Comando tradicional

Método moderno

```
git switch nombre-rama
```

Más intuitivo y seguro

Volver a la rama anterior

```
git switch -
```

Cambiar entre Ramas

Método clásico

```
git checkout nombre-rama
```

Comando tradicional

Método moderno

```
git switch nombre-rama
```

Más intuitivo y seguro

Volver a la rama anterior

```
git switch -
```

Importante

Antes de cambiar de rama, asegúrate de que tu **working directory** esté limpio o haz commit de tus cambios.

Eliminar Ramas

1. Eliminar rama fusionada (seguro)

```
git branch -d nombre-rama #Git verifica que la rama esté fusionada
```

Eliminar Ramas

1. Eliminar rama fusionada (seguro)

```
git branch -d nombre-rama #Git verifica que la rama esté fusionada
```

2. Eliminar rama forzosamente

```
git branch -D nombre-rama #Elimina aunque no esté fusionada.  ¡Cuidado!
```

Eliminar Ramas

1. Eliminar rama fusionada (seguro)

```
git branch -d nombre-rama #Git verifica que la rama esté fusionada
```

2. Eliminar rama forzosamente

```
git branch -D nombre-rama #Elimina aunque no esté fusionada.  ¡Cuidado!
```

Advertencia

No puedes eliminar la rama en la que estás actualmente. Cambia a otra rama primero.

Renombrar Ramas

Renombrar rama actual

```
git branch -m nuevo-nombre
```

Renombrar Ramas

Renombrar rama actual

```
git branch -m nuevo-nombre
```

Renombrar otra rama

```
git branch -m nombre-viejo nombre-nuevo
```

Renombrar Ramas

Renombrar rama actual

```
git branch -m nuevo-nombre
```

Renombrar otra rama

```
git branch -m nombre-viejo nombre-nuevo
```

Ejemplo práctico

- `git branch -m master main` — Actualizar nomenclatura
- `git branch -m feat/login feature/login` — Corregir formato

Ejemplo Práctico: Flujo Básico

Crear y trabajar en una rama

```
# Crear nueva rama para característica
git switch -c feature/calculadora

# Hacer cambios y commits
git add calculadora.py
git commit -m "feat:  implementar suma"

# Volver a main
git switch main

# Eliminar rama (después de fusionar)
git branch -d feature/calculadora
```

3

Estrategias de Ramas

¿Por qué Necesitamos Estrategias?

El Problema

Sin una estrategia clara, los equipos crean ramas sin control, generando caos y conflictos.

¿Por qué Necesitamos Estrategias?

El Problema

Sin una estrategia clara, los equipos crean ramas sin control, generando caos y conflictos.

Consecuencias de no tener estrategia:

- Nombres inconsistentes de ramas
- No se sabe qué está en producción
- Conflictos difíciles de resolver
- Código sin revisar en producción
- Dificultad para rastrear cambios

¿Por qué Necesitamos Estrategias?

El Problema

Sin una estrategia clara, los equipos crean ramas sin control, generando caos y conflictos.

Consecuencias de no tener estrategia:

- Nombres inconsistentes de ramas
- No se sabe qué está en producción
- Conflictos difíciles de resolver
- Código sin revisar en producción
- Dificultad para rastrear cambios

Solución: Adoptar una estrategia de ramas estándar

Estrategias Principales

Git Flow

Creado por: Vincent Driessen (2010)

Características:

- Múltiples ramas permanentes
- Estructura muy definida
- Ideal para releases planificados
- Más complejo pero robusto

Estrategias Principales

Git Flow

Creado por: Vincent Driessen (2010)

Características:

- Múltiples ramas permanentes
- Estructura muy definida
- Ideal para releases planificados
- Más complejo pero robusto

Feature Branch

También llamado: GitHub Flow

Características:

- Una rama principal (main)
- Ramas por característica
- Más simple y flexible
- Más propensa a errores

Ambas son válidas — la elección depende del proyecto

4

Git Flow

Git Flow: Estructura

Ramas Permanentes

- `main` — Código en producción, siempre estable
- `develop` — Integración de características para próximo release

Git Flow: Estructura

Ramas Permanentes

- `main` — Código en producción, siempre estable
- `develop` — Integración de características para próximo release

Ramas Temporales

- `feature/*` — Nuevas características (desde develop)
- `release/*` — Preparación de release (desde develop)
- `hotfix/*` — Correcciones urgentes (desde main)

Git Flow: Diagrama Visual

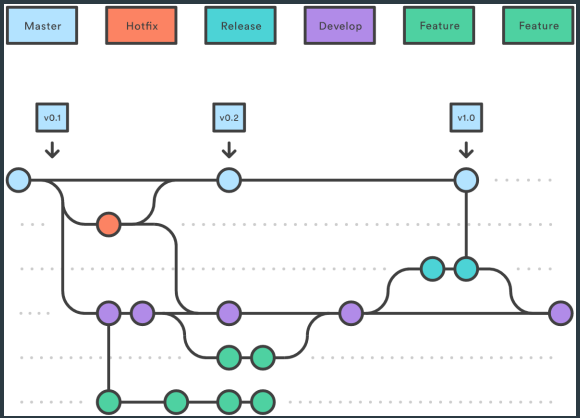


Figure: Fuente: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

Git Flow: Ciclo de Vida Feature

1 Crear feature desde develop

```
git switch develop
git switch -c feature/nueva-funcionalidad
```

Git Flow: Ciclo de Vida Feature

1. Crear feature desde develop

```
git switch develop
git switch -c feature/nueva-funcionalidad
```

2. Desarrollar y hacer commits

Git Flow: Ciclo de Vida Feature

1. Crear feature desde develop

```
git switch develop
git switch -c feature/nueva-funcionalidad
```

2. Desarrollar y hacer commits
3. Finalizar feature (fusionar a develop)

```
git switch develop
git merge feature/nueva-funcionalidad
git branch -d feature/nueva-funcionalidad
```

Git Flow: Ciclo de Vida Hotfix

1. Crear hotfix desde main

```
git switch main & git switch -c hotfix/corregir-bug-critico
```

Git Flow: Ciclo de Vida Hotfix

1. Crear hotfix desde main

```
git switch main & git switch -c hotfix/corregir-bug-critico
```

2. Corregir y commit

Git Flow: Ciclo de Vida Hotfix

1. Crear hotfix desde main

```
git switch main & git switch -c hotfix/corregir-bug-critico
```

2. Corregir y commit
3. Fusionar a main y develop

```
git switch main
git merge hotfix/corregir-bug-critico
git switch develop
git merge hotfix/corregir-bug-critico
git branch -d hotfix/corregir-bug-critico
```

Git Flow: El Addon git-flow-avh

git-flow (AVH Edition)

Extensión oficial que simplifica los comandos de Git Flow creada por **Peter van der Does**.

Git Flow: El Addon git-flow-avh

git-flow (AVH Edition)

Extensión oficial que simplifica los comandos de Git Flow creada por **Peter van der Does**.

Instalación

```
# Linux (Debian/Ubuntu)
sudo apt install git-flow

# macOS
brew install git-flow-avh

# Windows (con Chocolatey)
choco install git-flow-avh
```

git-flow: Inicialización

Inicializar Git Flow en repositorio

```
git flow init
```

git-flow: Inicialización

Inicializar Git Flow en repositorio

```
git flow init
```

Preguntas interactivas

```
Which branch should be used for bringing forth production releases?
Branch name for production releases:  [main]

Which branch should be used for integration of the "next release"?
Branch name for "next release" development:  [develop]

How to name your supporting branch prefixes?
Feature branches?  [feature/]
Release branches?  [release/]
Hotfix branches?  [hotfix/]
```

git-flow: Comandos Simplificados

Feature

```
# Iniciar
git flow feature start login

# Finalizar
git flow feature finish login
```

git-flow: Comandos Simplificados

Feature

```
# Iniciar
git flow feature start login

# Finalizar
git flow feature finish login
```

Hotfix

```
# Iniciar
git flow hotfix start 1.0.1

# Finalizar
git flow hotfix finish 1.0.1
```

git-flow: Comandos Simplificados

Feature

```
# Iniciar
git flow feature start login

# Finalizar
git flow feature finish login
```

Hotfix

```
# Iniciar
git flow hotfix start 1.0.1

# Finalizar
git flow hotfix finish 1.0.1
```

Release

```
git flow release start 2.0.0
git flow release finish 2.0.0
```

git-flow: Comparación de Comandos

Sin git-flow

```
git switch develop
git switch -c feature/login
# ...  trabajo ...
git switch develop
git merge feature/login
git branch -d feature/login
```

5 comandos

Con git-flow

```
git flow feature start login
# ...  trabajo ...
git flow feature finish login
```

2 comandos

git-flow: Comparación de Comandos

Sin git-flow

```
git switch develop
git switch -c feature/login
# ...  trabajo ...
git switch develop
git merge feature/login
git branch -d feature/login
```

5 comandos

Con git-flow

```
git flow feature start login
# ...  trabajo ...
git flow feature finish login
```

2 comandos

Ventaja

Menos errores, más eficiencia, nombres consistentes automáticamente.

Git Flow: Ventajas y Desventajas

Ventajas

- Muy estructurado y organizado
- Producción siempre estable (main)
- Ideal para releases planificados
- Hotfixes bien definidos
- Menos conflictos

Git Flow: Ventajas y Desventajas

Ventajas

- Muy estructurado y organizado
- Producción siempre estable (main)
- Ideal para releases planificados
- Hotfixes bien definidos
- Menos conflictos

Desventajas

- Más complejo de aprender
- Requiere disciplina del equipo
- No ideal para CI/CD continuo
- Overhead para proyectos pequeños
- Dos ramas permanentes

Git Flow: Ventajas y Desventajas

Ventajas

- Muy estructurado y organizado
- Producción siempre estable (main)
- Ideal para releases planificados
- Hotfixes bien definidos
- Menos conflictos

Desventajas

- Más complejo de aprender
- Requiere disciplina del equipo
- No ideal para CI/CD continuo
- Overhead para proyectos pequeños
- Dos ramas permanentes

Ideal para: Proyectos grandes, equipos medianos/grandes, releases programados

5

Feature Branch Work-flow

Feature Branch: Estructura

Concepto

Toda nueva característica o corrección se desarrolla en una rama separada desde `main`.

Feature Branch: Estructura

Concepto

Toda nueva característica o corrección se desarrolla en una rama separada desde `main`.

Reglas básicas:

- Una sola rama permanente: `main`
- Cada developer crea ramas para sus tareas
- Las ramas se fusionan a `main` cuando están listas
- `main` siempre debe estar desplegable

Feature Branch: Estructura

Concepto

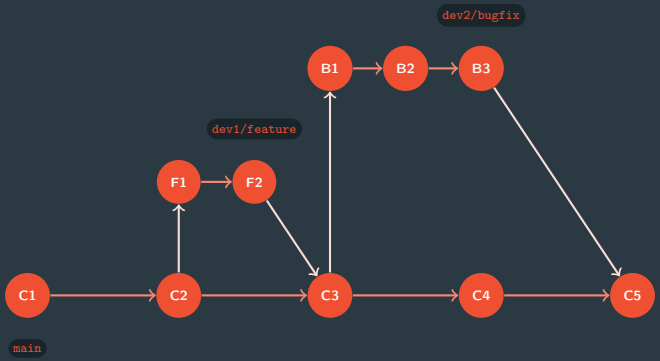
Toda nueva característica o corrección se desarrolla en una rama separada desde `main`.

Reglas básicas:

- Una sola rama permanente: `main`
- Cada developer crea ramas para sus tareas
- Las ramas se fusionan a `main` cuando están listas
- `main` siempre debe estar desplegable

También conocido como **GitHub Flow**

Feature Branch: Diagrama Visual



Cada desarrollador trabaja en sus propias ramas y las fusiona cuando completa

Feature Branch: Flujo de Trabajo

1 Crear rama desde main

```
git switch main
git pull
git switch -c juan/agregar-autenticacion
```

Feature Branch: Flujo de Trabajo

1. Crear rama desde main

```
git switch main
git pull
git switch -c juan/agregar-autenticacion
```

2. Desarrollar y hacer commits

Feature Branch: Flujo de Trabajo

1. Crear rama desde main

```
git switch main
git pull
git switch -c juan/agregar-autenticacion
```

2. Desarrollar y hacer commits
3. Fusionar a main

```
git switch main & git pull
git merge juan/agregar-autenticacion
git push
```

Feature Branch: Convenciones de Nombres

Formato común: desarrollador/descripción

- `juan/login-page`
- `maria/fix-database-error`
- `carlos/refactor-api`

Feature Branch: Convenciones de Nombres

Formato común: desarrollador/descripción

- `juan/login-page`
- `maria/fix-database-error`
- `carlos/refactor-api`

Alternativas:

- Por ticket: `JIRA-123-login`, `issue-45-bug`
- Mixto: `juan/feature/login`

Feature Branch: Convenciones de Nombres

Formato común: desarrollador/descripción

- `juan/login-page`
- `maria/fix-database-error`
- `carlos/refactor-api`

Alternativas:

- Por ticket: `JIRA-123-login`, `issue-45-bug`
- Mixto: `juan/feature/login`

Importante: El equipo debe acordar y seguir una convención

Feature Branch: Ventajas y Desventajas

Ventajas

- Muy simple de entender
- Ideal para CI/CD
- Flexibilidad total
- Rápido para equipos pequeños
- Menos overhead

Feature Branch: Ventajas y Desventajas

Ventajas

- Muy simple de entender
- Ideal para CI/CD
- Flexibilidad total
- Rápido para equipos pequeños
- Menos overhead

Desventajas

- Sin estructura formal
- Propenso a errores humanos
- Puede volverse caótico
- Difícil gestionar releases
- Requiere code reviews estrictas

Feature Branch: Ventajas y Desventajas

Ventajas

- Muy simple de entender
- Ideal para CI/CD
- Flexibilidad total
- Rápido para equipos pequeños
- Menos overhead

Desventajas

- Sin estructura formal
- Propenso a errores humanos
- Puede volverse caótico
- Difícil gestionar releases
- Requiere code reviews estrictas

Ideal para: Equipos pequeños, despliegue continuo, proyectos ágiles

6

Comparación y Buenas Prácticas

¿Cuál Elegir?

Elige Git Flow si:

- Tienes releases programados
- Equipo mediano o grande
- Múltiples versiones en paralelo
- Necesitas estabilidad máxima
- Proceso formal de QA

¿Cuál Elegir?

Elige Git Flow si:

- Tienes releases programados
- Equipo mediano o grande
- Múltiples versiones en paralelo
- Necesitas estabilidad máxima
- Proceso formal de QA

Elige Feature Branch si:

- Despliegue continuo
- Equipo pequeño
- Proyecto ágil/startup
- Quieres simplicidad de desarrollo
- Menos burocracia en el flujo de trabajo

¿Cuál Elegir?

Elige Git Flow si:

- Tienes releases programados
- Equipo mediano o grande
- Múltiples versiones en paralelo
- Necesitas estabilidad máxima
- Proceso formal de QA

Elige Feature Branch si:

- Despliegue continuo
- Equipo pequeño
- Proyecto ágil/startup
- Quieres simplicidad de desarrollo
- Menos burocracia en el flujo de trabajo

No hay una respuesta única — depende del contexto

Buenas Prácticas Universales

Aplica a cualquier estrategia

- **Nombres descriptivos:** `feature/login-oauth` mejor que `rama1`
- **Ramas cortas:** Fusionar frecuentemente para evitar conflictos
- **main siempre estable:** No hacer commit directo a main
- **Sincronizar:** Hacer `git pull` antes de crear ramas
- **Eliminar ramas fusionadas:** Mantener repositorio limpio

7

¿Qué sigue?

Fusionar Ramas: Merge

Próxima Clase

Hemos visto cómo crear y gestionar ramas, pero ¿cómo las **unimos**?

Fusionar Ramas: Merge

Próxima Clase

Hemos visto cómo crear y gestionar ramas, pero ¿cómo las **unimos**?

En la Clase 5 aprenderemos:

- `git merge` — Fusionar ramas
- `git rebase` — Reescribir historial
- Resolución de conflictos
- Fast-forward vs 3-way merge
- Estrategias de fusión
- Cherry-pick

¡La verdadera magia de Git viene al combinar ramas!

> *Branches are cheap and easy, use them early and often.*
>
> — Git Best Practices

Comandos Clave

Git Estándar

```
git branch
git switch -c nombre
git switch nombre
git branch -d nombre
git branch --merged
```

git-flow

```
git flow init
git flow feature start
git flow feature finish
git flow hotfix start
git flow hotfix finish
```

¡Gracias!

Nos vemos en la próxima clase