

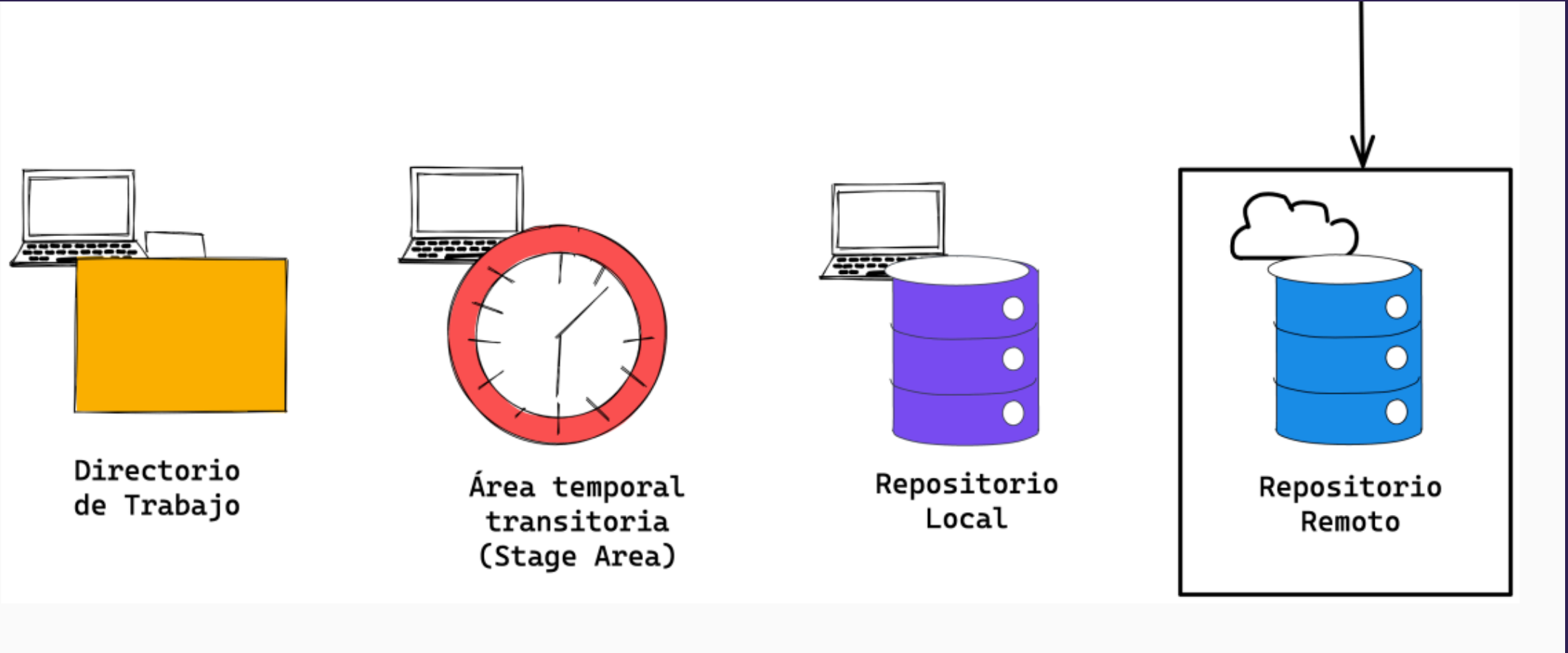
# Git Básico

---

**Arturo Silvelo**

Try New Roads

# Trabajando de Forma Remota



# Configuración en GitHub

---

## Creando una clave SSH

---

### 1. Generar clave SSH

```
ssh-keygen -t ed25519 -C "tu-email@ejemplo.com"
```

### 2. Copiar la clave pública

```
cat ~/.ssh/id_ed25519.pub
```

### 3. Agregar en GitHub: Settings → SSH and GPG keys → New SSH key

## SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

## Authentication keys



SSH

## S12400

SHA256:c7hd/eMDUhjrUayczMNj8m1vo0w8XEthNi1q1kk0kc/4

Added on Jul 13, 2022

Last used within the last 2 months — Read/write

Delete



SSH

## HP Oficina

SHA256:qwt f3yHvSpk9FDaj2WXIS00n0GjZG5dvme9Bb1FLtV0

Added on Jun 5, 2023

Last used within the last 7 months — Read/write

Delete

## Add new SSH Key

## Title

## Key type

Authentication Key ↕

## Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

## ¿Qué es SSH?

---

**SSH (Secure Shell)** es un protocolo de red seguro que permite establecer una comunicación encriptada entre cliente y servidor. Proporciona autenticación mediante verificación de identidad sin necesidad de contraseñas repetitivas, protegiendo todos los datos durante la transmisión y garantizando la seguridad de las conexiones remotas.

## ¿Por qué usar SSH con Git?

---

- **Sin contraseñas repetitivas:** Una vez configurado, no pide credenciales
- **Mayor seguridad:** Autenticación por criptografía de clave pública
- **Mejor para automatización:** Scripts y CI/CD sin problemas de autenticación
- **Recomendado por GitHub:** Método preferido para desarrolladores



## Cómo funciona SSH

---

- **Clave Privada** → Se queda en tu computadora (secreta)
- **Clave Pública** → Se sube a GitHub (se puede compartir)
- **Autenticación** → GitHub verifica que tienes la clave privada
- **Conexión Segura** → Comunicación cifrada establecida

# Crear un Repositorio

---

## Crear repositorio en Github

---


1. **Nuevo repositorio:** Click en "New repository" o el botón "+"
2. **Nombre:** Elegir nombre descriptivo para el proyecto
3. **Descripción:** Breve explicación del propósito del repositorio
4. **Visibilidad:** Seleccionar Public (público) o Private (privado)
5. **Inicialización:** Opcionalmente agregar README.md, .gitignore o licencia

## Create a new repository

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


*Required fields are marked with an asterisk (\*).*


Owner \*      Repository name \*

 silvelo /

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

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** ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)


Choose a license


License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Grant your Marketplace apps access to this repository

You are subscribed to 1 Marketplace app.

☐  **Travis CI**  
Test and deploy with confidence

 You are creating a public repository in your personal account.

[Create repository](#)

# Clonar un Repositorio

---

## Desde GitHub a local

---

Para clonar un repositorio que ya ha sido creado:

- Usando SSH

```
git clone git@github.com:usuario/repositorio.git
```

- Usando HTTPS

```
git clone https://github.com/usuario/repositorio.git
```

Al clonar un repositorio `git` crea una carpeta con el mismo nombre del repositorio.

Para especificar un nombre diferente para la carpeta local:

```
git clone git@github.com:usuario/repositorio.git mi-carpeta
```

## Enlazar un Repositorio Local

---

Si tenemos un repositorio local y lo queremos enlazar a uno remoto:

```
git remote add origin git@github.com:usuario/repositorio.git
```



## ¿Qué es `origin` ?

- Es un **nombre o alias** que le damos al repositorio remoto
- Por **convención** se usa "origin" para el repositorio principal
- Puedes usar cualquier nombre: `upstream`, `backup`, `produccion`, etc.

## ¿Qué es la dirección?

- Es la **URL** donde está alojado el repositorio remoto
- Formato SSH: `git@github.com:usuario/repositorio.git`
- Formato HTTPS: `https://github.com/usuario/repositorio.git`

## Verificar remotos configurados

---

Para verificar los repositorios configurados

```
git remote -v
```

### Salida típica:

```
origin  git@github.com:usuario/repositorio.git (fetch)  
origin  git@github.com:usuario/repositorio.git (push)
```

# Trabajar con Cambios

---

## Subir cambios al remoto

---

- **Primera vez (establecer tracking):**

```
git push -u origin main  
# Rama local 'main' → Rama remota 'main'
```

- **Primera vez nombres distintos:**

```
git push -u origin mi-rama:rama-remota  
# Rama local 'mi-rama' → Rama remota 'rama-remota'
```

- **Después, simplemente:**

```
git push
```

## ¿Por qué `-u` la primera vez?

---

El parámetro `-u` (o `--set-upstream`) establece una **relación de seguimiento** entre tu rama local y la rama remota.

**Sin `-u`:**

- Tienes que especificar siempre: `git push origin main`
- Git no sabe a dónde hacer push por defecto

**Con `-u`:**

- Git "recuerda" la rama remota
- Puedes usar solo `git push` en el futuro

## Estados del tracking

---

- Rama sin tracking:

```
git push origin main # Siempre necesario especificar
```

- Rama con tracking establecido:

```
git push # Git sabe a dónde enviar
```

- **Ver estado de tracking:**

```
git branch -vv
```

- **Salida ejemplo:**

```
* main      a1b2c3d [origin/main] Último commit  
feature f4e5d6c [origin/desarrollo: ahead 2] Commits pendientes  
local      g7h8i9j Sin tracking configurado
```

- **Interpretación:**

- `main` vinculada con `origin/main`
- `feature` vinculada con `origin/desarrollo` (2 commits por subir)
- `local` sin vinculación remota

# Traer Cambios del Remoto

---



## Git Fetch - Solo descargar

---

```
git fetch origin
```

### ¿Qué hace?

- Descarga cambios del repositorio remoto
- **NO modifica** tu trabajo local
- Actualiza las referencias remotas ( `origin/main` , `origin/develop` )
- Te permite revisar antes de fusionar

## Git Pull - Descargar y fusionar

---

```
git pull origin main
```

### ¿Qué hace?

- Hace `git fetch` + `git merge` automáticamente
- Descarga cambios Y los fusiona en tu rama actual
- Más rápido pero menos control
- Puede crear conflictos inmediatamente

## ¿Cuándo usar cada uno?

---

- **Usar Git Fetch cuando:**

- Quieres revisar los cambios antes de fusionar
- Trabajas en equipo y necesitas ser cauteloso
- Quieres evitar conflictos inesperados
- Necesitas comparar cambios primero

- **Usar Git Pull cuando:**

- Confías en los cambios remotos
- Trabajas solo en la rama
- Quieres sincronizar rápidamente
- No hay riesgo de conflictos importantes

## Resolver conflictos en Pull

---

Si hay conflictos al hacer `git pull`:

### 1. Git marca los archivos con conflictos

```
<<<<<<< HEAD
Tu código local
=====
Código del remoto
>>>>>>> origin/main
```

**2. Editar y resolver conflictos manualmente**

**3. Agregar archivos resueltos**

```
git add archivo-resuelto.txt
```

**4. Completar el merge**

```
git commit
```

# Ramas en Remoto

---

## Crear y subir nueva rama

---

- **Crear rama local (opción tradicional):**

```
git checkout -b feature/nueva-funcionalidad
```

- **Crear rama local (opción moderna):**

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

- **Primera subida con tracking:**

```
git push -u origin feature/nueva-funcionalidad
```

- **Siguientes pushes:**

```
git push # Ya no necesita especificar origin ni rama
```



## Trabajar con ramas remotas existentes

---

- Con checkout (tradicional):

```
git checkout -b feature/login origin/feature/login
```

- Con switch (moderno):

```
git switch -c feature/login origin/feature/login
```

- O más simple (Git moderno):

```
git switch feature/login  
# Git automáticamente crea tracking con origin/feature/login
```

## Eliminar Ramas Remotas

---

- Eliminar rama remota:

```
git push origin --delete feature/rama-antigua
```

- O usando sintaxis corta:

```
git push origin :feature/rama-antigua
```

## Fusionar Ramas

---

- **Cambiar a rama destino:**

```
git switch main
```

- **Fusionar rama de feature:**

```
git merge feature/nueva-funcionalidad
```

- **Subir cambios fusionados:**

```
git push origin main
```

# Pull Request

---

## ¿Qué es un Pull Request?

---

Un **Pull Request (PR)** es una solicitud para fusionar cambios de una rama a otra, típicamente desde una rama de feature hacia la rama principal.

### Propósito:

- Solicitar fusión de cambios
- Permitir revisión de código por el equipo
- Facilitar discusión sobre los cambios
- Mantener historial de decisiones

## ¿Por qué usar Pull Requests?

---

- **Revisión de código:** Otros desarrolladores revisan antes de fusionar
- **Control de calidad:** Evita que código problemático llegue a main
- **Discusión:** Comentarios y sugerencias en línea
- **Historial:** Registro de qué cambios se hicieron y por qué
- **Tests automáticos:** CI/CD puede ejecutar pruebas antes del merge

# Flujo básico de Pull Request

---

1. Crear rama de feature
2. Hacer commits con cambios
3. Push de la rama al remoto
4. Crear Pull Request en GitHub
5. Revisión y discusión
6. Merge a rama principal
7. Limpieza (eliminar rama)

## 1. Crear rama de feature

---

```
git checkout main  
git pull origin main  
git checkout -b feature/login-usuario
```

### Buenas prácticas:

- Partir siempre de la rama principal actualizada
- Nombres descriptivos: `feature/`, `bugfix/`, `hotfix/`
- Una funcionalidad por rama



## 2. Hacer commits

---

```
# Trabajar en la funcionalidad
git add .
git commit -m "Agregar formulario de login"

git add .
git commit -m "Validar datos de usuario"

git add .
git commit -m "Integrar autenticación JWT"
```

### Consejos:

- Commits pequeños y específicos
- Mensajes descriptivos
- Funcionalidad completa y probada

### 3. Push al remoto

---

```
git push -u origin feature/login-usuario
```

**Primera vez:** Usar `-u` para establecer tracking

**Siguientes pushes:**

```
git push
```

## 4. Crear PR en GitHub

---

1. **GitHub detecta la nueva rama** y sugiere crear PR
2. **O manual:** "New pull request" → Seleccionar ramas
3. **Completar información:**
  - Título descriptivo
  - Descripción detallada
  - Reviewers (revisores)
  - Labels, Assignees, etc.

## 5. Revisión y discusión

---

### Los revisores pueden:

- Comentar líneas específicas del código
- Sugerir cambios
- Aprobar o rechazar
- Solicitar modificaciones

### El autor puede:

- Responder comentarios
- Hacer commits adicionales
- Resolver discusiones

## 6. Merge del Pull Request

---

### Opciones de merge en GitHub:

- **Merge commit:** Crea commit de merge (preserva historial)
- **Squash and merge:** Combina todos los commits en uno
- **Rebase and merge:** Reaplica commits sin merge commit

## 7. Limpieza después del merge

---

```
# Cambiar a main y actualizar
git switch main
git pull origin main

# Eliminar rama local
git branch -d feature/login-usuario

# GitHub elimina automáticamente la rama remota
```

## Estados de un Pull Request

---

- **Open:** En proceso, esperando revisión
- **Draft:** Borrador, trabajo en progreso
- **Ready for review:** Listo para revisión
- **Changes requested:** Necesita modificaciones
- **Approved:** Aprobado, listo para merge
- **Merged:** Fusionado exitosamente
- **Closed:** Cerrado sin fusionar

## Template para descripción de PR

---

### ## ¿Qué hace este PR?

Breve descripción de los cambios

### ## ¿Por qué es necesario?

Contexto y justificación

### ## ¿Cómo probarlo?

Pasos para verificar la funcionalidad

### ## Checklist

- [ ] Tests actualizados
- [ ] Documentación actualizada
- [ ] Sin conflictos con main
- [ ] Probado localmente



# FORK

---

## ¿Qué es un Fork?

---

Un **Fork** es una copia independiente de un repositorio que se crea en tu cuenta de GitHub.

### Características:

- **Copia completa** del repositorio original con todo su historial
- **En tu cuenta** de GitHub, tienes control total
- **Independiente** del repositorio original
- **Mantiene conexión** para sincronizar cambios

## ¿Cuándo usar Fork?

---

- **Contribuir a proyectos de terceros** donde no tienes permisos de escritura
- **Experimentar** con código sin afectar el proyecto original
- **Crear tu propia versión** de un proyecto open source
- **Aprender** estudiando el código de otros proyectos
- **Colaborar** en proyectos donde no eres colaborador directo

## Fork vs Clone vs Branch

Acción	Fork	Clone	Branch
Ubicación	GitHub → Tu GitHub	Remoto → Local	Local
Propósito	Contribuir a terceros	Trabajar localmente	Features/experimentos
Permisos	Tu propio repositorio	Solo lectura/escritura según permisos	Mismo repositorio

# Flujo de Trabajo con Fork

---

## 1. Fork del repositorio original

---

1. Ve al repositorio que quieres hacer fork
2. Click en "**Fork**" (esquina superior derecha)
3. Selecciona tu cuenta como destino
4. GitHub crea la copia en tu cuenta

## 2. Clonar tu fork

---

```
git clone git@github.com:tu-usuario/repositorio.git  
cd repositorio
```

**Importante:** Clonas TU fork, no el repositorio original.

### 3. Agregar upstream

---

```
git remote add upstream git@github.com:usuario-original/repositorio.git
```

#### Verificar remotos:

```
git remote -v
```

#### Resultado:

```
origin      git@github.com:tu-usuario/repositorio.git (fetch)
origin      git@github.com:tu-usuario/repositorio.git (push)
upstream    git@github.com:usuario-original/repositorio.git (fetch)
upstream    git@github.com:usuario-original/repositorio.git (push)
```



## 4. Mantener Fork Actualizado

---

**Traer cambios del repositorio original:**

```
git fetch upstream  
git switch main  
git merge upstream/main  
git push origin main
```

**O usando pull directamente:**

```
git pull upstream main  
git push origin main
```

## 5. Crear feature y hacer cambios

---

```
# Crear rama de feature
git switch -c feature/nueva-funcionalidad

# Hacer cambios y commits
git add .
git commit -m "feat: Agregar nueva funcionalidad"

# Subir a TU fork
git push -u origin feature/nueva-funcionalidad
```

## 6. Crear Pull Request al original

---

1. Ve a TU fork en GitHub
2. GitHub detecta la nueva rama y sugiere PR
3. **Importante:** Verificar que el PR va hacia el repositorio original
4. Completar descripción del PR
5. Enviar para revisión

# Sincronización Avanzada

---

## Resolver conflictos con upstream

---

Si el repositorio original ha cambiado mientras trabajabas:

```
# Actualizar main desde upstream
git fetch upstream
git switch main
git merge upstream/main

# Rebase tu feature sobre main actualizado
git switch feature/nueva-funcionalidad
git rebase main
```

## Eliminar fork

---

Si ya no necesitas el fork:

1. Ve a tu fork en GitHub
2. **Settings** → **General**
3. Bajar hasta "**Danger Zone**"
4. "**Delete this repository**"
5. Escribir el nombre del repositorio para confirmar

# Buenas Prácticas con Fork

---

## Antes de hacer Fork

---

- **Lee la documentación** del proyecto
- **Revisa las guidelines** de contribución
- **Busca issues** etiquetados como "good first issue"
- **Verifica** que el proyecto esté activo



## Trabajando con Fork

---

- **Mantén actualizado** tu fork regularmente
- **Una rama** por feature o bugfix
- **Commits pequeños** y descriptivos
- **Prueba localmente** antes del PR
- **Sigue las convenciones** del proyecto original

## Pull Requests desde Fork

---

- **Título claro** describiendo el cambio
- **Descripción detallada** con contexto
- **Referencias** a issues relacionados
- **Screenshots** si hay cambios visuales
- **Tests** si el proyecto los requiere

# Diferencias Fork vs Colaborador

---

## Como Colaborador (con permisos)

---

```
# Clonar directamente el repositorio original
git clone git@github.com:empresa/proyecto.git

# Crear rama de feature
git switch -c feature/nueva-funcionalidad

# Push directo al repositorio original
git push -u origin feature/nueva-funcionalidad
```

## Como Contributor (via Fork)

---

```
# Clonar TU fork
git clone git@github.com:tu-usuario/proyecto.git

# Agregar upstream
git remote add upstream git@github.com:empresa/proyecto.git

# Push a tu fork, PR al original
git push -u origin feature/nueva-funcionalidad
```

## Ventajas del modelo Fork

---

- **Seguridad:** Los mantenedores controlan qué se fusiona
- **Calidad:** Todo código pasa por revisión
- **Historial limpio:** Solo cambios aprobados llegan al main
- **Escalabilidad:** Miles de contribuidores sin permisos directos
- **Aprendizaje:** Los nuevos pueden practicar sin riesgo