Control de Versiones
Git

11 de Marzo de 2016

Germán Scillone

Índice

Control de Versiones

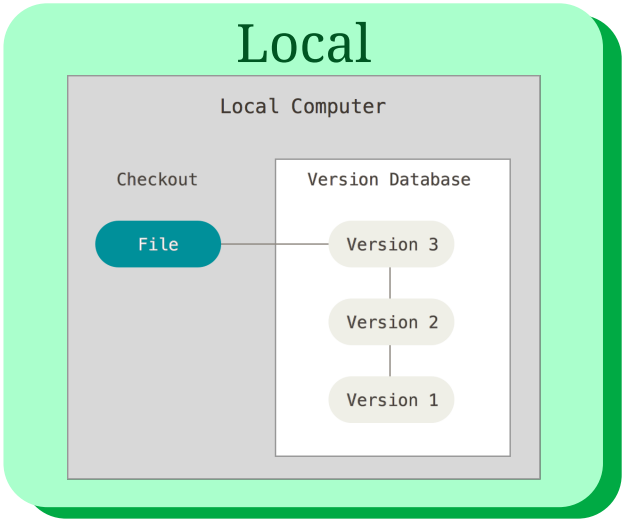
¿Qué es?

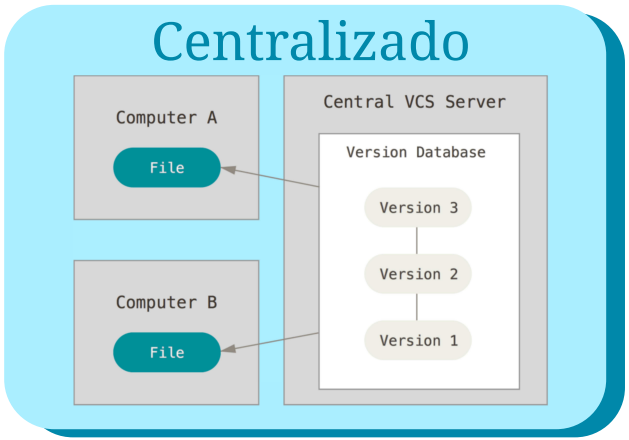
El control de versiones es un sistema que almacena y organiza los cambios realizados a uno o varios archivos para permitir acceder a distintas versiones de los mismos posteriormente.

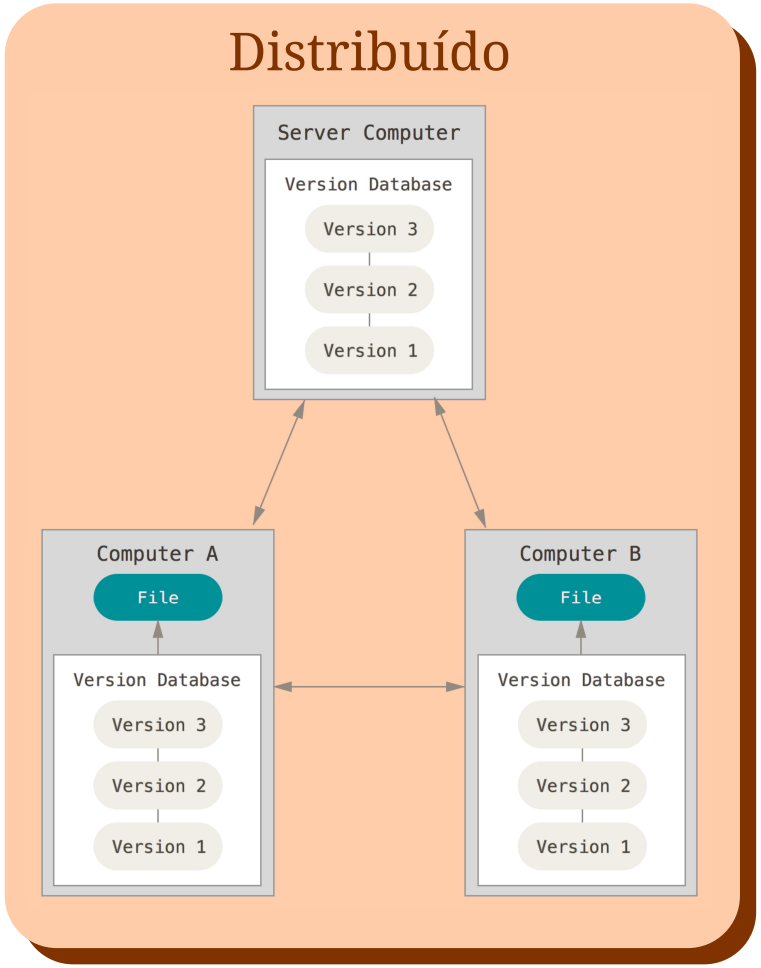
Ventajas:

- Permite revertir archivos o el proyecto entero a versiones anteriores.
- Permite comparar versiones, ver cambios.
- Permite trabajar de forma colectiva en grupos de desarrollo.
- Permite recuperar los archivos en caso de pérdida o de algún error catastrófico.

Tipos de Sistemas de Control de Versiones







git en General

Git se desarrollo en 2005 para servir de sistema de control de versiones en el desarrollo del kernel de Linux.

Las premisas principales del diseño fueron:

- Velocidad.
- Diseño sencillo.
- Soporte para desarrollos no lineales (infinitas ramas paralelas).
- Completamente Distribuido.
- Capaz de manejar grandes proyectos (velocidad y tamaño de archivos).

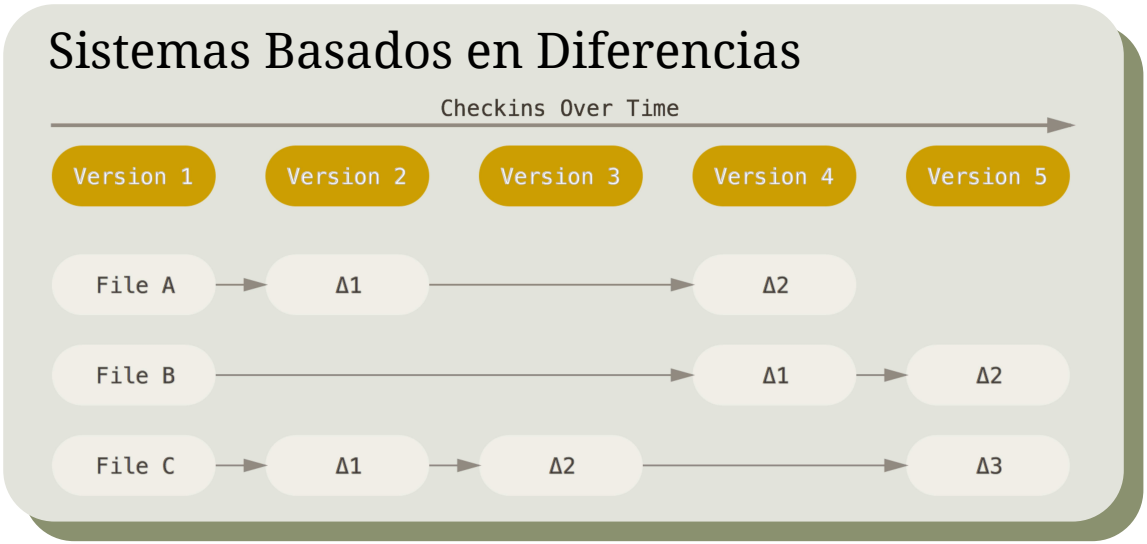
Git: Principios Fundamentales

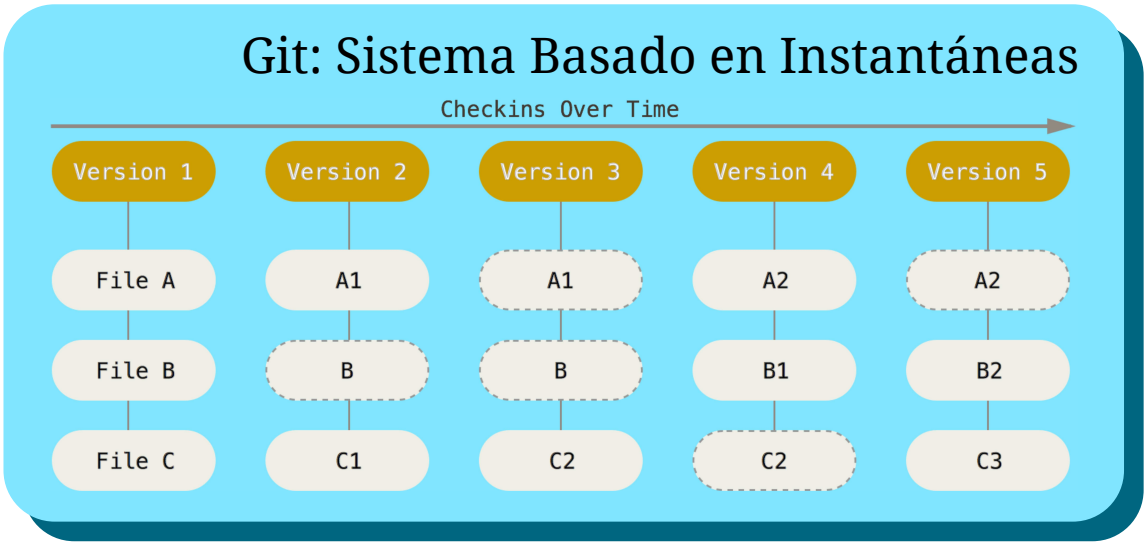
- Todos los cambios son locales.
- Siempre agrega datos.
- Control de integridad - Checksum SHA-1 hash, que además sirve de identificación de las versiones.

SHA-1 Hash

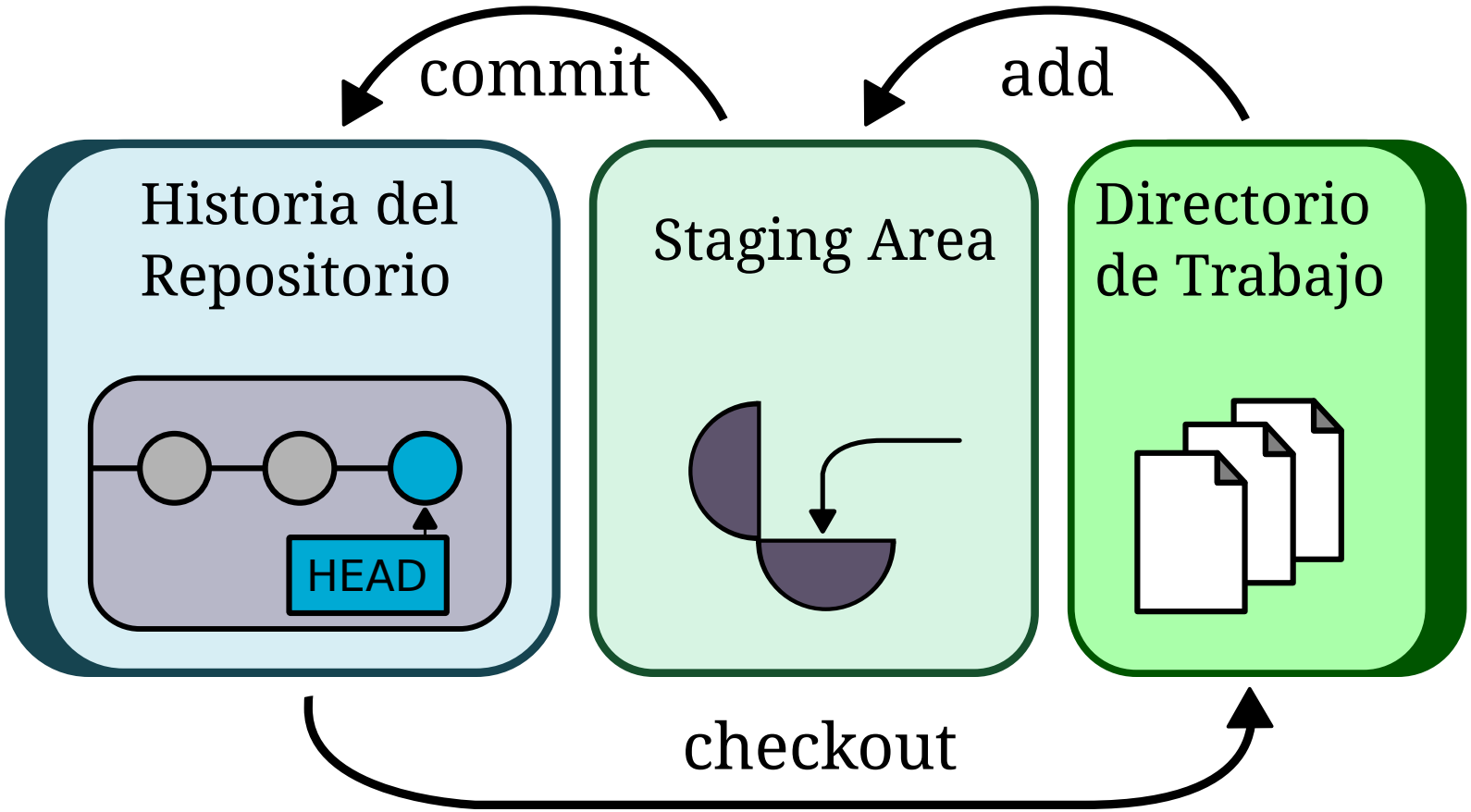
37b9ac6f52152487da426b52f8697cd6d8b06373

git: Principios Fundamentales





Las Tres Secciones



Instalación

Descarga

Sitio Oficial | https://git-scm.com/

Linux (Debian / Ubuntu)

```
>$ apt-get install git
```

Windows

Descargar Instalador

Next/ Next/ Accept / Next
Next/ Next/ Next / Finish

Ejecutar git bash

Configuración Inicial

git config

Este comando se utiliza para ver y modificar la configuración de git. Para empezar a usar este control de versiones es necesario configurar dos parámetros particulares: Nombre de Usuario y Mail de Usuario.

- git config --global user.name 'German Sc'
- git config --global user.mail 'germanscillone@gmail.com'

git init / git clone

git init

Crea un repositorio local en el directorio de trabajo. Básicamente una carpeta oculta llamada ".git" que contiene toda la información que necesita para funcionar.

git clone <url>

Sirve para obtener una copia de un repositorio existente (accesible por <url>) en el directorio actual de trabajo.

git status

git status

Este comando es fundamental para tener un panorama del estado actual del proyecto respecto a la última versión.

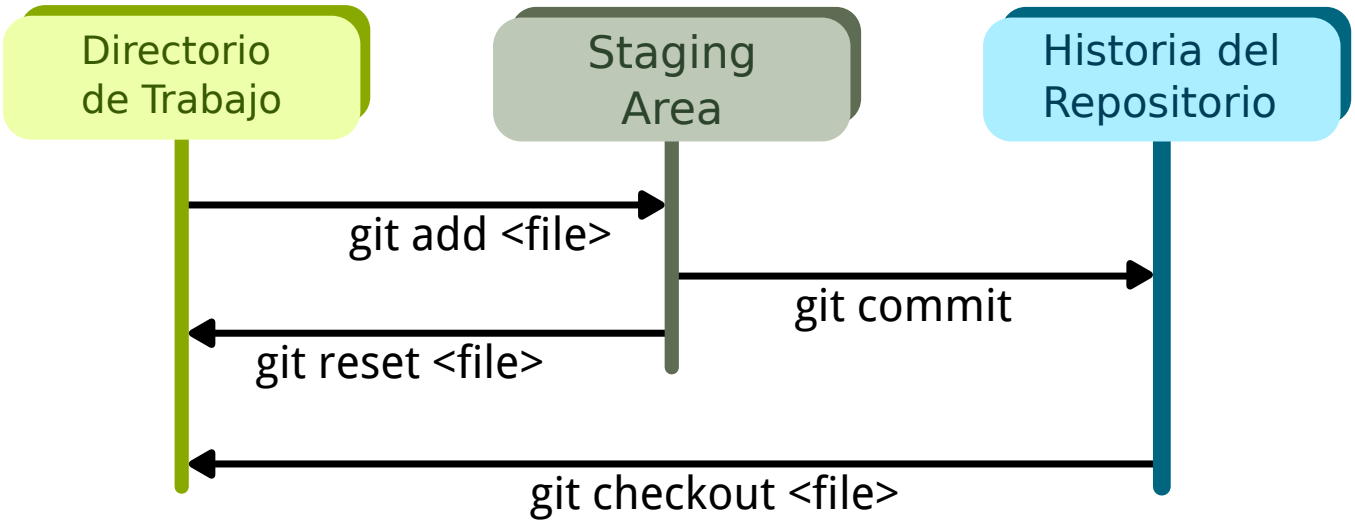
Compara los archivos actuales con las versiones del ultimo commit e indica:

- Que archivos fueron modificados, creados o eliminados.
- Cuales de esos cambios están agregados para el próximo commit.
- Comandos recomendados a utilizar para modificar el estado presentado.

git add

git add <archivo>

Este comando permite elegir qué archivos (modificados) queremos agregar a la próxima versión que guardemos en el repositorio.



git commit

git commit -m ”<mensaje>”

El comando commit crea una instantánea del proyecto con los archivos y cambios agregados a la ”staging area”, acompañados con un mensaje.
Cualquier archivo modificado que no haya sido agregado al commit quedará modificado en el disco, y aparecerá como cambio que puede agregarse al próximo commit.

git diff

git diff

Al igual que el anterior, presenta los archivos que cambiaron en el directorio respecto al último commit, pero además muestra cuáles fueron esos cambios imprimiendo las dos versiones de los archivos linea por linea.

```
diff --git a/src/main.c b/src/main.c
index 3ffea1a..f7ac387 100644
--- a/src/main.c
+++ b/src/main.c
@@ -3,9 +3,9 @@

 int main ( int argc, char* argv[] )
 {
-        if (argc != 1)
+        if (argc > 2)
         {
-                printf("Hola Mundo!\n");
+                printf("Chau Luna?\n");

         } else {
```

git log

git log

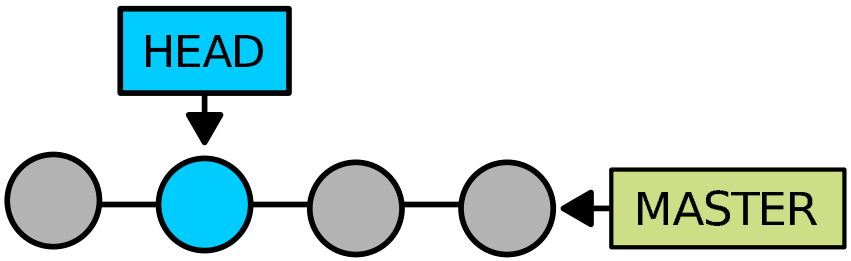
Muestra la historia de commits en el repositorio.
A partir de diversas opciones permite configurar completamente la información que proporcionada, pudiendo mostrar:

- Los cambios (diff) realizados a los archivos de cada commit.
- Filtrar commits por autor, por fecha, por período, etc...
- Graficar la linea temporal de commits.
- Mostrar información reducida o ampliada.
- Combinar las opciones anteriores y más.

git checkout <commit> <file>

git checkout

El comando permite cambiar el directorio de trabajo al estado de cualquier <commit> anterior.
Esta acción es segura, ya que es de solo-lectura, se pueden modificar archivos y experimentar, e incluso hacer nuevos commits, pero estos no sobreescribirán ni afectarán a la rama de trabajo.



git revert / git reset

git revert <commit>

Este comando deshace los cambios de introducidos en <commit>, pero mantiene la historia. Esto lo logra mediante la creación de un nuevo commit que revierte los cambios que se hicieron en <commit>.

git reset [--soft --mixed --hard] <commit>

Permite revertir la rama actual a cualquier punto anterior de la historia del repositorio. De este modo, permite además deshacer commits definitivamente, borrándolos de la HISTORIA.

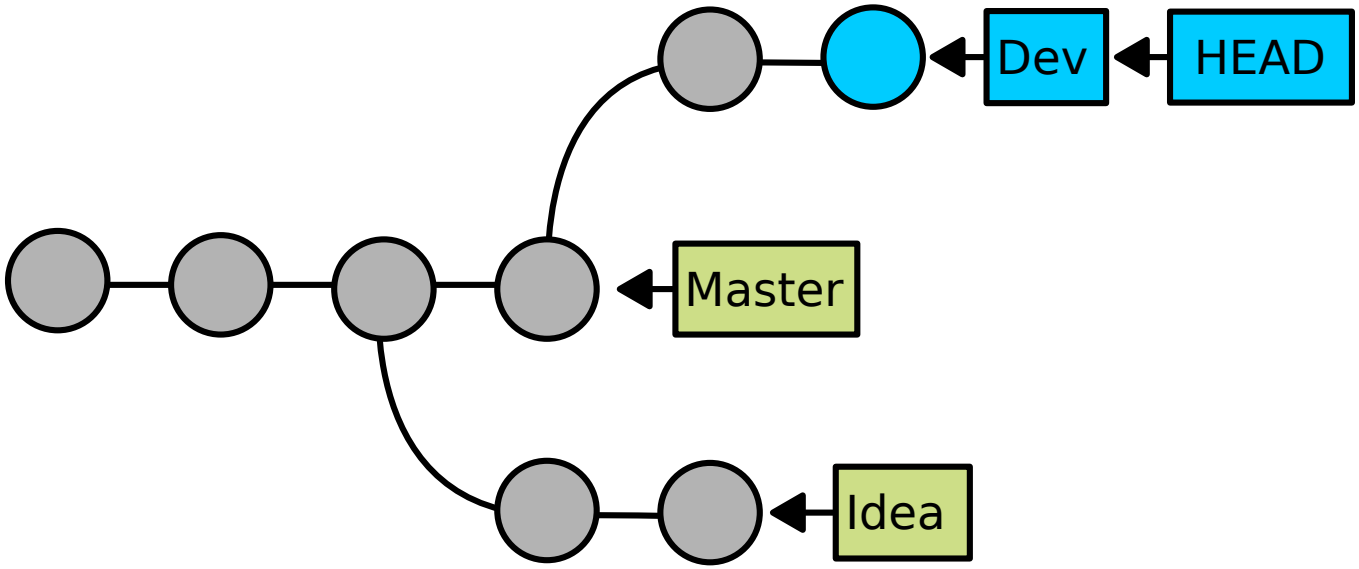
PRECAUCIÓN

git reset es uno de los pocos comandos con los que se puede perder información de manera irreversible!

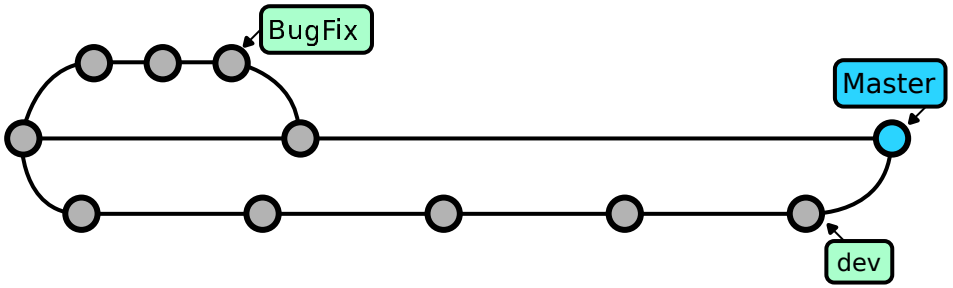
Conceptos de Branches

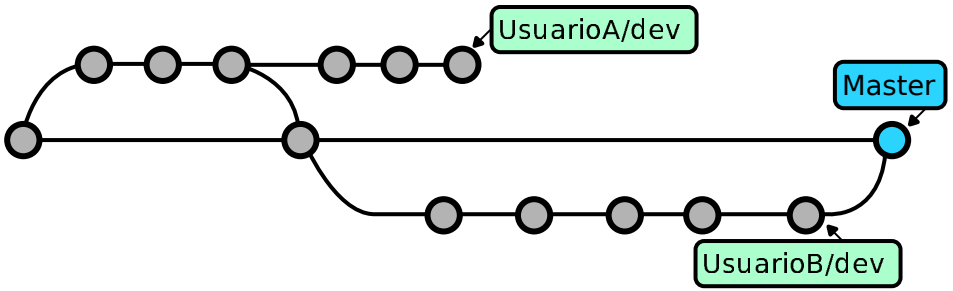
Las ramas de desarrollo son una de las características más destacadas de git.

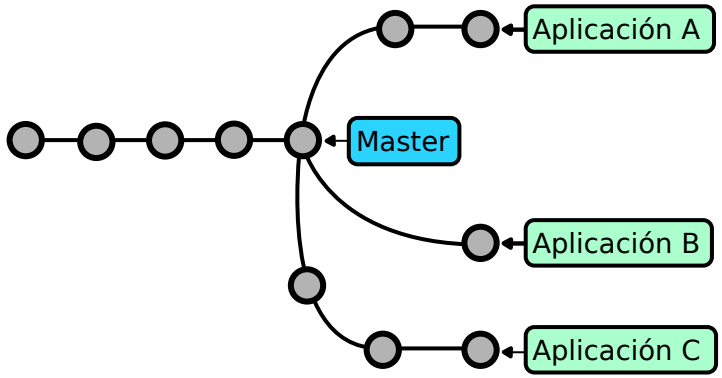
Con total simplicidad se pueden utilizar para solicitar un directorio de trabajo nuevo, basado en cualquier commit de la historia, e independiente de cualquier otra rama de desarrollo.



Flujos de Trabajo y Usos de las Ramas







Creación de branches

git branch <nombre>

El comando git branch se utiliza para la administración de las ramas del proyecto, permitiendo crear, renombrar o eliminar ramas, pero no realiza el cambio del directorio de trabajo entre ramas.

git checkout <nombre>

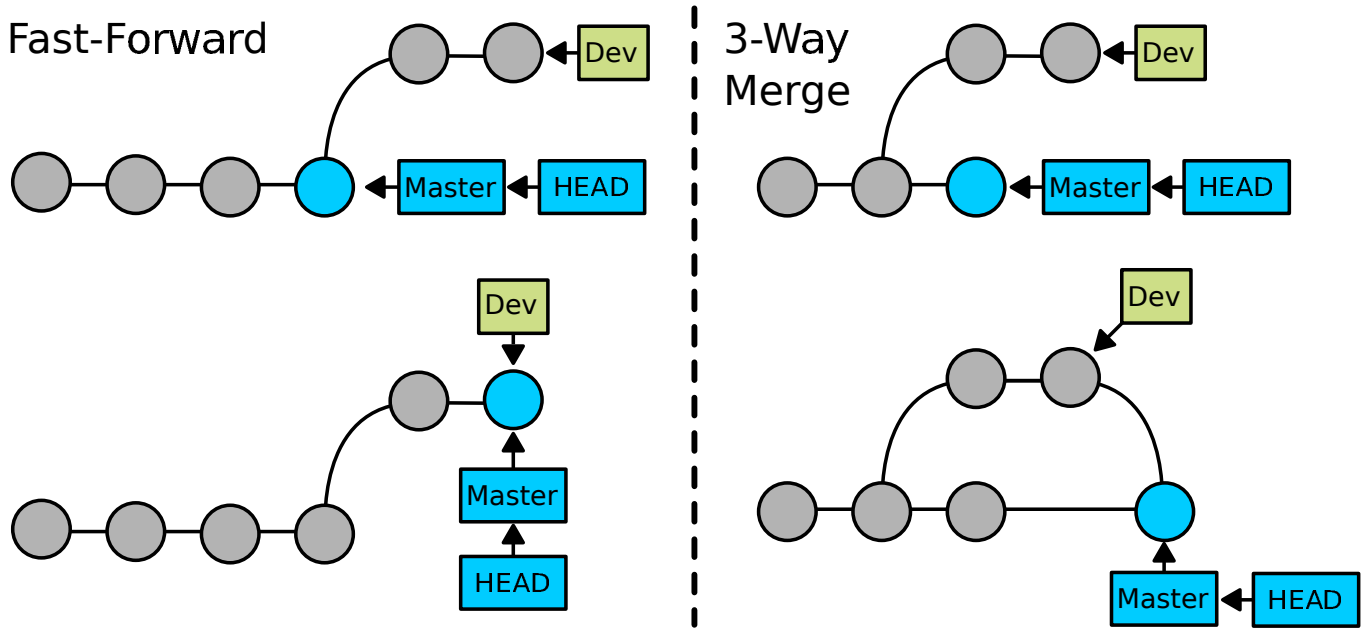
El comando checkout es el que se utiliza para cambiar el directorio de trabajo entre las versiones de las distintas ramas.

Además, mediante la opción ”-b” permite crear las ramas directamente y cambiar a un commit.

Merge

git merge <branch>

El comando permite fusionar la rama <branch> con la rama de trabajo. Siempre que pueda git realizará un fast-forward merge, salvo que se indique lo contrario con la opción ”–no-ff”.



Resolución de conflictos

En caso que se intenten fusionar dos ramas que modificaron las mismas partes de uno o más archivos, git no decidirá qué versión es la correcta e indicará que existe un conflicto antes de completar el merge commit.

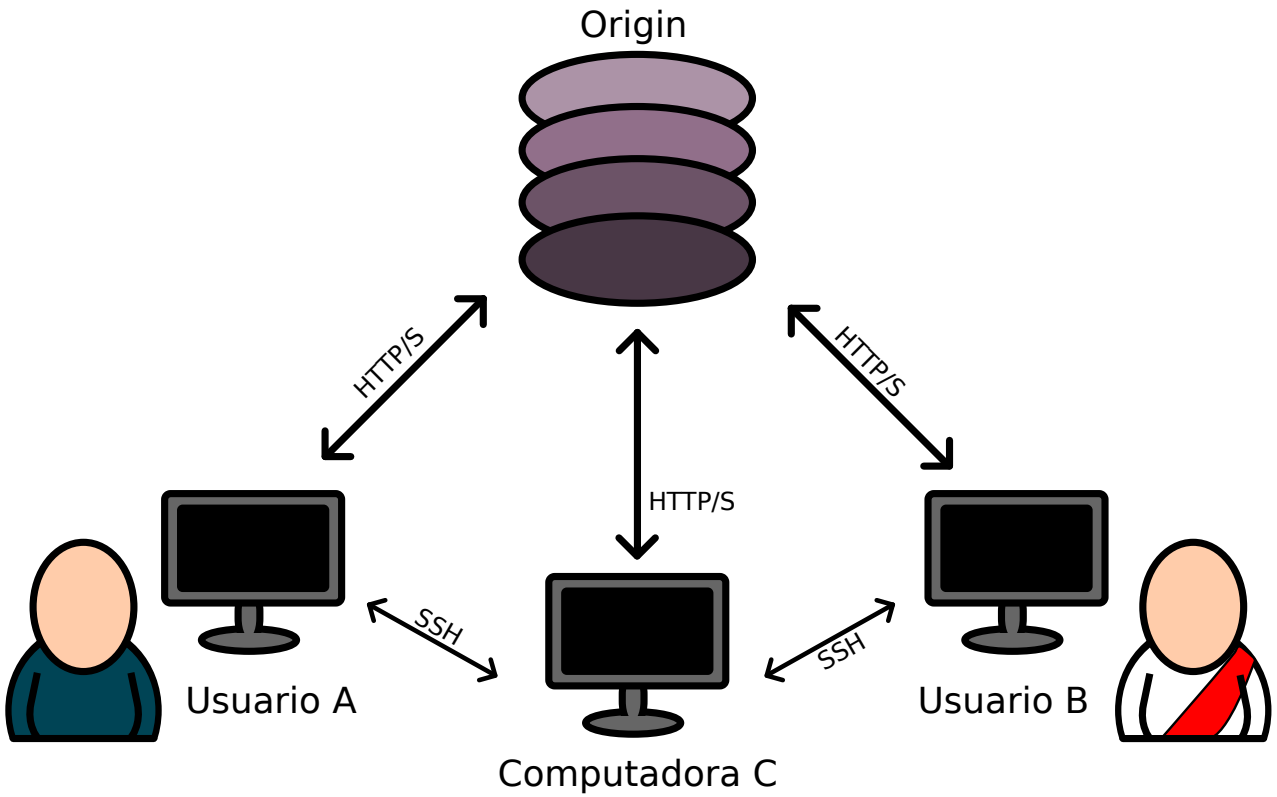
Los pasos a seguir para solucionar el problema son los siguientes:

- *git status* (lista los archivos conflictivos)
- Se accede a los archivos y se opta por una de las modificaciones (o se hace una nueva)
- *git add <archivos>*
- *git commit -m "<mensaje>"*

Otra ventaja de git es que los comandos utilizados para solucionar estos conflictos son los mismos que los que se utilizan normalmente, por lo que no altera el flujo de trabajo.

Remotes en git

Git permite establecer conexiones con diversos repositorios remotos (repositorios hosteados en internet o en una red privada) llamados remotes que sirven tanto como back-up del proyecto así como para compartirlo entre personas o computadoras.



GitHub / GitLab y Otras Yerbas

GitLab

Atlassian Bitbucket

¿Qué Son?

Son servicios de hosting para repositorios online. Además permiten interactuar de manera directa con el repositorio, modificando archivos, agregando, eliminando, viendo la historia de commits y los cambios en los mismos, todo a través de una interfaz gráfica más amigable que la consola de comandos.

Administración de Remotes

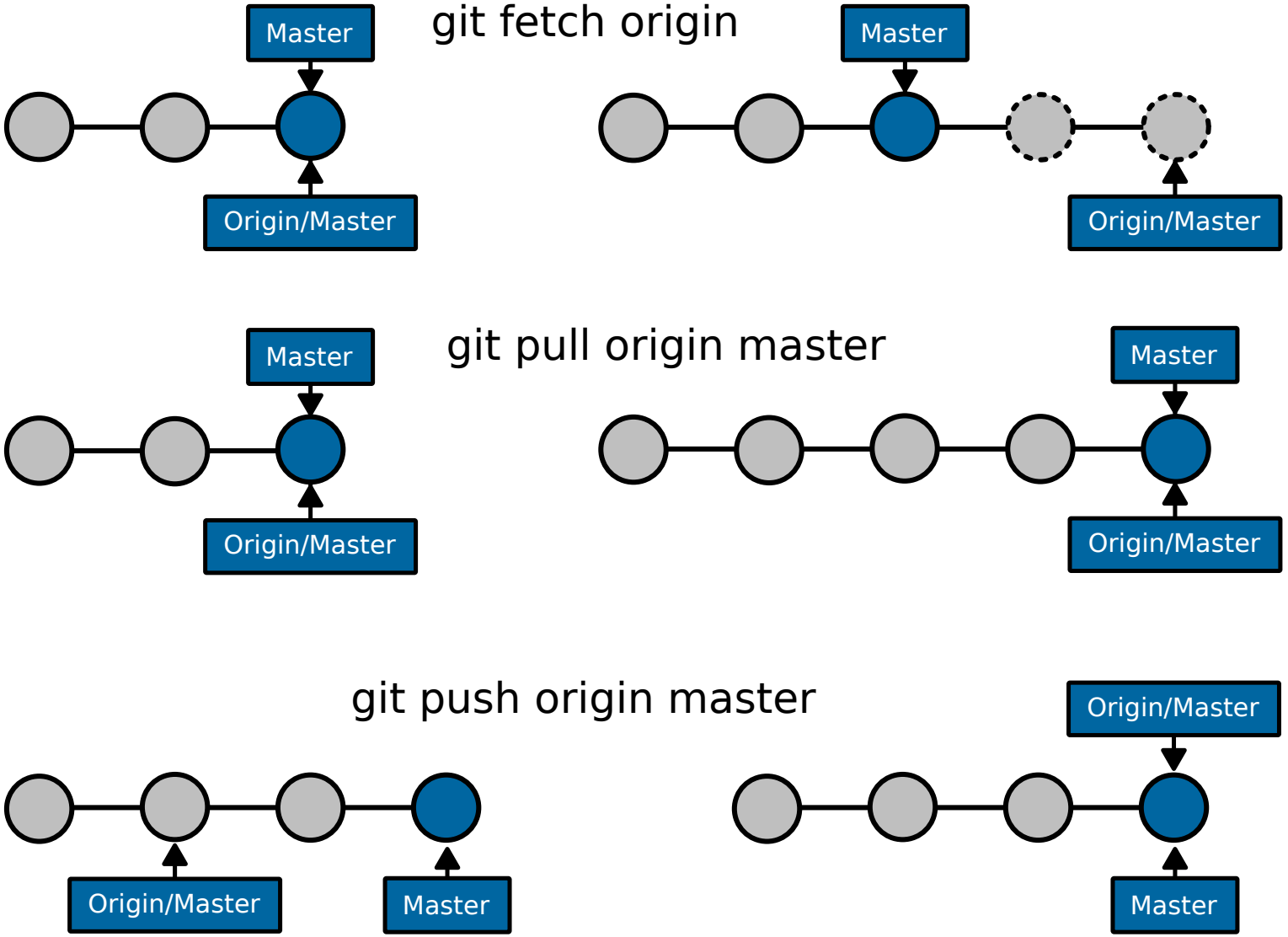
git remote

Del mismo modo que git branch, git remote es el comando que se utiliza para administrar las conexiones a repositorios remotos.

- git remote add <nombre> <url>
- git remote rm <nombre>
- git remote rename <nombre> <nuevo nombre>

Al clonar un repositorio automáticamente se crea un remote llamado *origin* que redirecciona a la url de donde se clonó.

git fetch / pull / push



Recomendaciones

Algunas consideraciones para trabajar de forma prolija y aprovechar al máximo las características del control de versiones:

- Usar ramas a mansalva.
- Se trackean archivos fuente y de configuración, se ignoran los archivos generados (ejecutables, archivos objeto, log, pdf, etc).
- Todos los archivos deben ser trackeados o ignorados.
- Mensajes de commits claros.
- Commits cortos y específicos. No implementar (o arreglar) varias cosas en un único commit.
- Todo código en la rama ”Master” debe compilar.

Tutoriales

Hay infinidad de tutoriales en la web sobre control de versiones, algunos destacados son:

- Atlassian: `https://www.atlassian.com/git/tutorials`
- Git-Tower: `https://www.git-tower.com/learn/`
- GitHub: `https://try.github.io/`

Git-Tower provee además una hoja con los comandos más comunes, y para qué se usan a modo de machete para tener a mano, además de otra hoja con buenas prácticas similar a la sección anterior, entre otros recursos interesantes.

DESCARGAR ZIP

Bibliografía

- Chacon, Scott & Straub, Ben. *"Pro Git"*.
- Tobias Günther.*"Learn Version Control With Git"*.
- `https://www.gitref.org/`
- `https://www.git-scm.com/`