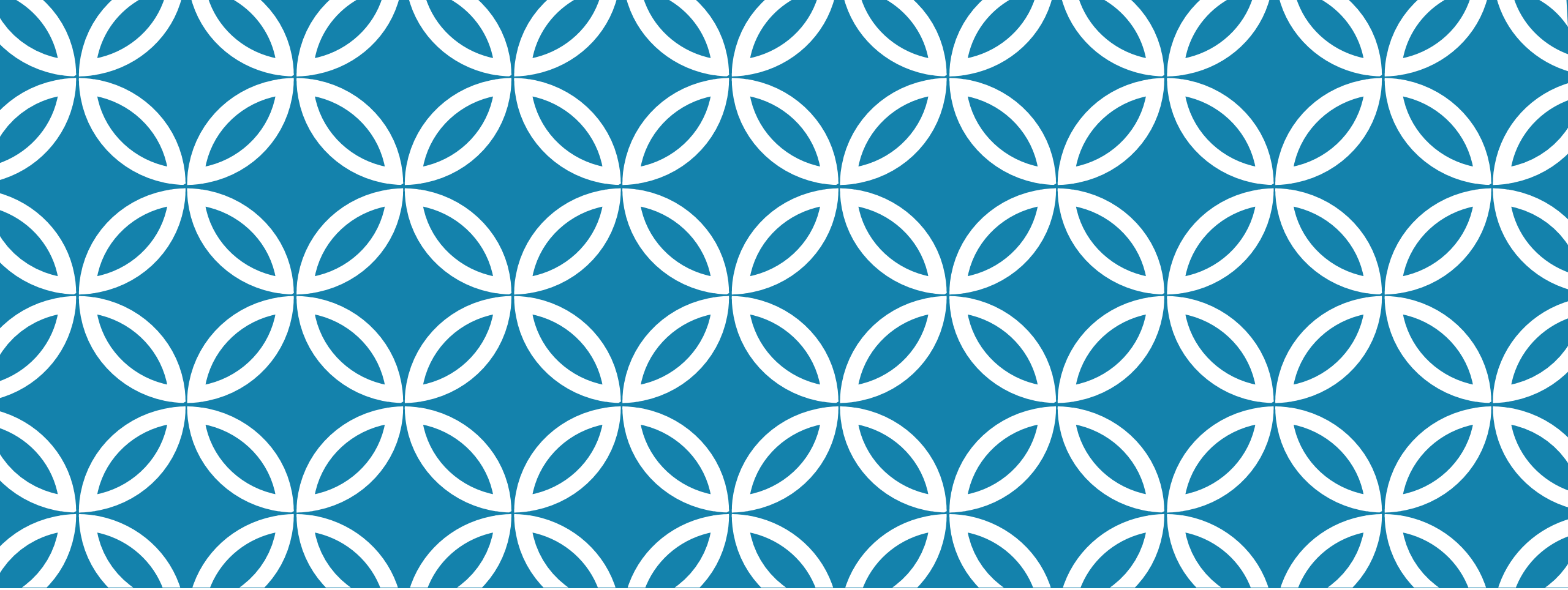


GIT

Control de versiones

AGENDA

- Control de versiones
- GIT
- Flujo de trabajo de GIT
- Comandos Básicos

CONTROL DE VERSIONES

- También conocido como "*control de código fuente*" es la práctica de rastrear y gestionar los cambios en el código de software.

- Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo.

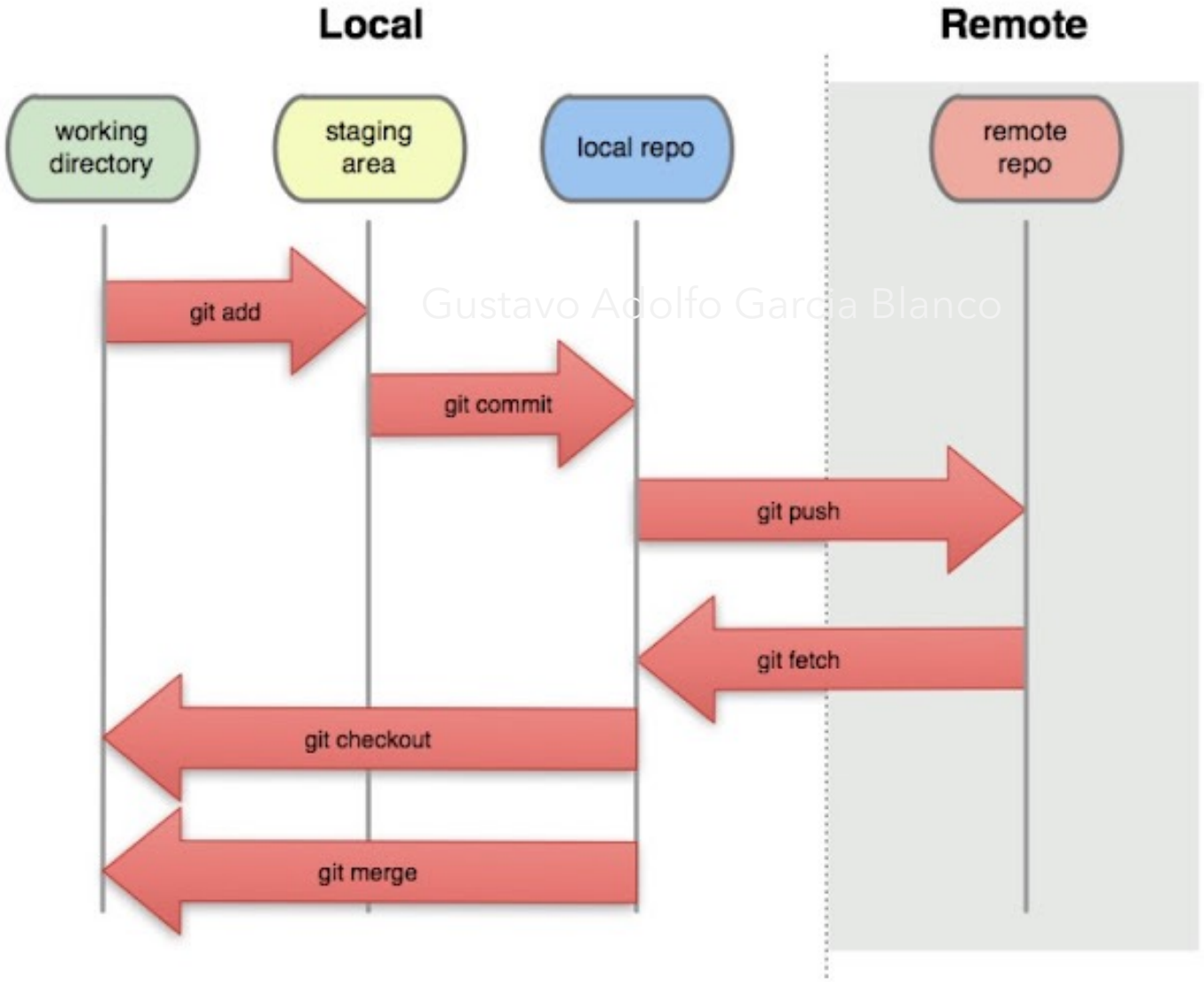
- El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos.

GIT

- GIT es un VCS *distribuido* gratuito y de código abierto. Es el sistema de control de versiones moderno más utilizado en el mundo, desarrollado por Linus Torvalds, el famoso creaor del kernel del sistema operativo Linux, en 2005.

- GIT, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido). En lugar de tener un único espacio de manera habitual como en los sistemas de control de versiones de antaño; CVS o Subversion, en GIT, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

FLUJO DE TRABAJO DE GIT



CONFIG

Este comando permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git. Estas variables pueden almacenarse en tres sitios distintos:

CONFIG

1. `/etc/gitconfig` contiene valores para todos los usuarios del sistema y todos sus repositorios. Si pasas la opción `--system` a `git config`, lee y escribe específicamente en este archivo.
2. `~/.gitconfig` este archivo es específico de tu usuario. Puede hacer que Git lea y escriba específicamente en este archivo pasando la opción `--global`.
3. `config` en el directorio de Git (es decir, `.git/config`) del repositorio que estés utilizando actualmente: Este archivo es específico del repositorio actual.

TU IDENTIDAD - CONFIG

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los `commits` de Git usan esta información, y es introducida de manera inmutable en los `commits` que envías:


```
$ git config --global user.name "your name"

$ git config --global user.email example@domain.com
```

TU IDENTIDAD

CONFIG

De nuevo, sólo necesitas hacer esto una vez si especificas la ópcion `-- global`, ya que Git siempre usará esta información para todo lo que hagas en este sistema. Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando de la diapostiva anterior sin la opción `--global` cuando éstes en ese proyecto.

COMPROBANDO TU CONFIGURACION

Si quieres comprobar tu configuración, puedes usar el comando:

- `git config --list`

Para mostrar todas las propiedades que Git ha configurado.

Puede que veas claves repetidas, porque Git lee la misma clave de distintos archivos (`/etc/gitconfig` y `~/.gitconfig` por ejemplo). En estos casos, Git usa el último valor para cada clave única que ve.

COMPROBANDO TU CONFIGURACION

También puedes comprobar el valor de que Git utilizará para una clave específica ejecutando `git config <key>:`


```
$ git config user.name
```

INIT

Este comando crea un repositorio Git vacío, básicamente un directorio `.git` con subdirectorios para `objects`, `refs/heads`, `refs/tags`, y archivos de plantilla. Se creará una rama inicial sin ningún `commit`.

REMOTE

El comando `git remote` te permite crear, ver y eliminar conexiones con otros repositorios. Las conexiones remotas se asemejan más a marcadores que a enlaces directos con otros repositorios. En lugar de brindar acceso en tiempo real a otro repositorio, funcionan como nombres prácticos que pueden emplearse para hacer referencia auna URL no tan sencilla.

REMOTE

El comando `git remote` es, en esencia, una interfaz para gestionar una lista de entradas remotas almacenadas en el archivo `./.git/config` del repositorio. Para ver el estado actual de la lista remota, se utilizan los siguientes comandos:

REMOTE

- `git remote` enumera las conexiones que tienes con otros repositorios.

- `git remote -v` realiza lo mismo que el comando anterior, pero incluye la URL de cada conexión.

REMOTE

- `git remote add <name> <url>` crea una nueva conexión a un repositorio remoto. Después de agregar un control remoto, podrá usar `<name>` como un atajo para `<url>` en otros comandos Git.

- `git remote rm <name>` elimina la conexión al repositorio remoto llamado `<name>`.

BRANCH

Una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación. Puedes concebirlas como una forma de solicitar un nuevo directorio de trabajo, un nuevo entorno de ensayo o un nuevo historial de proyecto. Las nuevas confirmaciones se registran en el historial de la rama actual, lo que crea una bifurcación en el historial del proyecto.

BRANCH

El comando **`git branch`** te permite crear, enumerar y eliminar ramas, así como cambiar su nombre. No te permite cambiar entre ramas o volver a unir un historial bifurcado. Por este motivo, **`git branch`** está estrechamente integrado con los comandos **`git checkout`** y **`git merge.`**

OPCIONES COMUNES BRANCH

- `git branch` enumera las ramas de tu repositorio. Es similar a `git branch --list`.

- `git branch <branch>` crea una nueva rama.

- `git branch -d <branch>` elimina una rama.

OPCIONES COMUNES BRANCH

- `git branch -D <branch>` fuerza la eliminación de la rama especificada, incluso si tiene cambios sin fusionar.

- `git branch -m <branch>` cambia el nombre de la rama actual a `<branch>`.

- `git branch -a` enumera todas las ramas remotas.

CHECKOUT

El comando `git checkout` opera sobre tres entidades distintas: archivos, configuraciones y ramas.

Los cambios de rama se asemejan al cambio de confirmaciones y archivos antiguos, ya que el directorio de trabajo se actualiza para reflejar la rama o revisión seleccionada. Sin embargo, los cambios nuevos se guardan en el historial del proyecto, es decir, no se trata de una operación de solo lectura.

CHECKOUT

El comando `git checkout` te permite desplazarte entre las ramas creadas por `git branch.` Al extraer una rama, se actualizan los archivos en el directorio de trabajo para reflejar la versión almacenada en esa rama y se indica a Git que registre todas las confirmaciones nuevas en dicha rama. Puedes contemplar todo esto como una forma de seleccionar la línea de desarrollo en la que trabajas.

CHECKOUT

- `git checkout <branch>` cambia de la rama actual en la que se esta trabajando a la `<branch>` especificada.

- `git checkout -b <new-branch>` se crea y se extrae la rama `<new-branch>` simultáneamente. La opción `-b` es una marca muy útil con la que Git ejecuta `git branch` antes de hacer lo propio con `git checkout <new-branch>.`

STATUS

El comando `git status` muestra el estado del directorio de trabajo y del área del entorno de ensayo. Permite ver los cambios que se han preparado, los que no y los archivos en los que Git no va a realizar el seguimiento. El resultado del estado no muestra ninguna información relativa al historial del proyecto. Para ello, debes usar `git log.`

STATUS

El comando `git status` es un comando relativamente sencillo. Simplemente, muestra lo que ha ocurrido con los comandos `git add` y `git commit`.

ADD

El comando `git add` añade un cambio del directorio de trabajo en el entorno de ensayo. De este modo, indica a Git que quieres incluir actualizaciones en un archivo concreto en la proxima confirmación. Sin embargo, `git add` no afecta al repositorio de manera significativa: en realidad, los cambios no se registran hasta que ejecutes `git commit.`

OPCIONES COMUNES ADD

- `git add <file>` prepara todos los cambios de `<file>` para la siguiente confirmación.

- `git branch <directory>` prepara los cambios de `<directory>` para la siguiente confirmación.

OPCIONES COMUNES ADD

- `git add -p` inicia una sesión de entorno de ensayo interactive que te permite elegir las partes de un archivo que quieres añadir a la siguiente confirmación.

- `git add .` prepara todos los cambios de la rama extraida para la siguiente confirmación.

COMMIT

Las confimaciones se crean con el comando `git commit` para capturar el estado de un proyecto en ese determinado momento. Las instantáneas de Git siempre se confirman en el repositorio local.

OPCIONES COMUNES COMMIT

- `git commit` confirma la instantánea preparado. El comando abrirá un editor de texto que te pedirá un mensaje para la confirmación.

- `git add -a` confirma una instantánea de todos los cambios del directorio de trabajo. Esta acción solo incluye las modificaciones a los archivos con seguimiento (los que se han añadido con `git add` en algún punto de su historial).

FETCH

El comando `git fetch` descarga commits, archivos y referencias de un repositorio remoto a tu repositorio local. Esta acción la llevas a cabo cuando quieres ver en qué han estado trabajando los demás. Este comando descarga el contenido remoto, pero no actualiza el estado de trabajo del repositorio local, por lo que tu trabajo actual no se verá afectado.

COMANDOS FETCH

- `git fetch <remote>` recupera todas las ramas del repositorio. También descarga todos los commits y archivos requeridos del otro repositorio.

- `git fetch <remote> <branch>` realiza la misma acción que el comando anterior, pero solo recupera la rama especificada.

OPCIONES FETCH

- `git fetch --all` una opción potente que recupera todos los repositorios remotos registrados y sus ramas.

- `git fetch --dry-run` ejecutará una demo del comando. Genera ejemplos de acciones que realizará durante la recuperación, pero no los aplica.

PULL

El comando `git pull` se emplea para extraer y descargar contenido desde un repositorio remoto y actualizer al instante el repositorio local para reflejar ese contenido. La fusión de cambios remotos de nivel superior en tu repositorio local es una tarea habitual de los flujos de trabajo de colaboración basados en Git. El comando `git pull` es, en realidad, una combinación, de dos comandos, `git fetch` y `git merge.`

OPCIONES PULL

- `git pull <remote>` recupera la copia del origen remoto especificado de la rama actual y se fusiona de inmediato en la copia local.

- `git pull --no-commit <remote>` de manera similar a la invocación predeterminada, extrae el contenido remoto, pero no crea una nueva confirmación de fusión.

OPCIONES PULL

- `git pull --rebase <remote>` al igual que en la anterior incorporación de cambios, en lugar de utilizar `git merge` para integrar la rama remota en la local, usa `git rebase.`

- `git pull --verbose` proporciona una salida detallada durante una incorporación de cambios que muestra el contenido descargado y los detalles de la fusión.

PUSH

El comando `git push` se usa para cargar contenido del repositorio local a un repositorio remoto. El envío es la forma de transferir confirmaciones desde tu repositorio local a un repositorio remoto. Es el equivalente a `git fetch`, pero mientras que al recuperar se importan las confirmaciones a ramas locales, al enviar estas se exportan a ramas remotas. Las ramas remotas se configuran mediante el comando `git remote.`

OPCIONES PUSH

- `git push <remote> <branch>` envía la rama especificada a `<branch>`, junto con todos los commits y objetos internos necesarios. De este modo se crea una rama local en el repositorio de destino.

- `git push <remote> --force` es igual que el comando anterior, pero fuerza el envío incluso si el resultado es una fusión sin avance rápido. No usarlo a menos de que tengas absoluta certeza de lo que estás haciendo.

OPCIONES PUSH

- `git push <remote> --all` envía todas tus ramas locales a una rama remota especificada.

- `git push <remote> --tags` las etiquetas no se envían automáticamente cuando envías una rama o usas la opción `--all.` La marca `--tags` envía todas las etiquetas locales al repositorio remoto.