GIT

Sistema de Control de Versionado (VCS)

GIT -

1. Que es un VCS y que tipos existen?
2. Git – Historia
3. Caracteristicas
    1. Integridad
    2. Estados
    3. Cyclo de vida.
4. Git – Instalación y comandos básicos.
5. GitHub.com

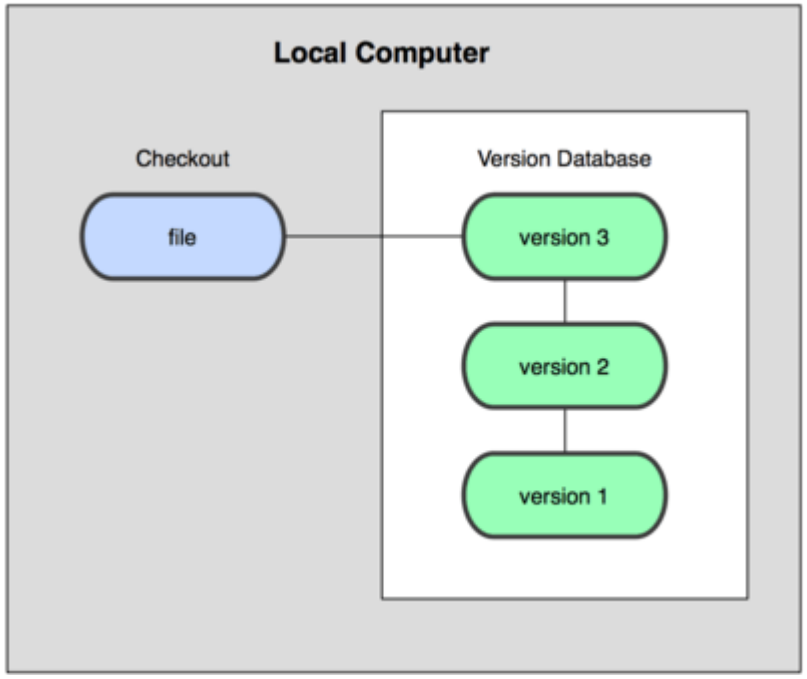
Que es un VCS.

- Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.
- No solo es útil para trabajar con código, sino también para llevar versiones de imágenes, libros, etc.

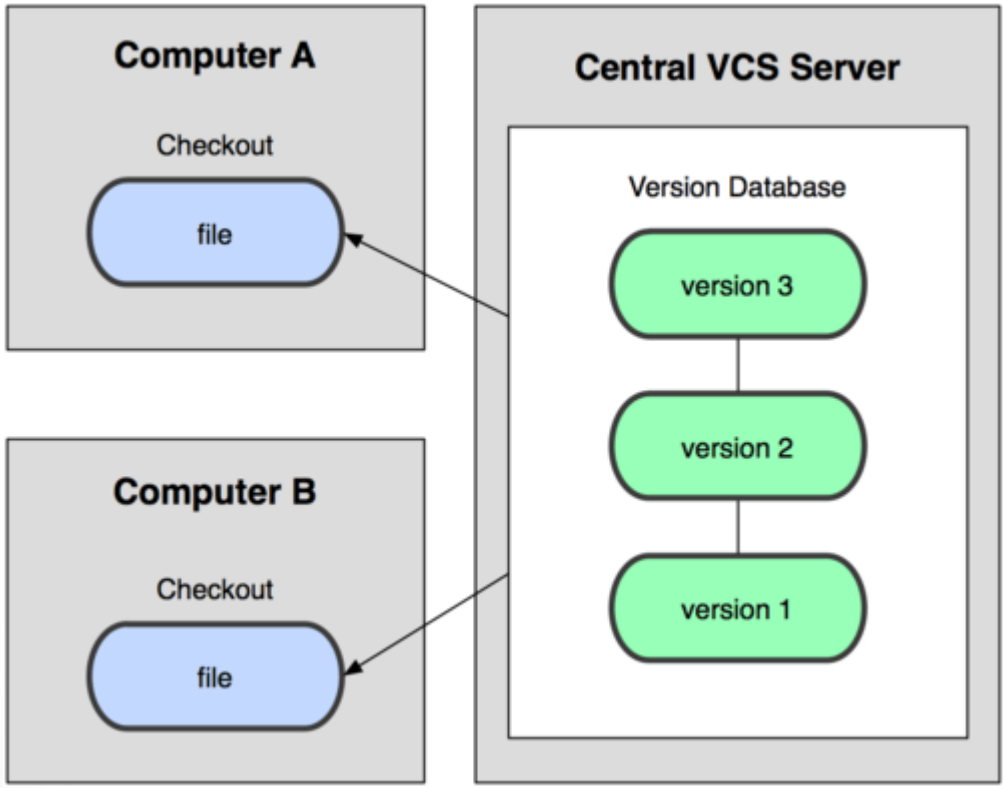
- Existen 3 tipos de VCS.
  - Locales.
  - Centralizados. (CVCS)
  - Distribuidos. (DVCS)

VCS - Local

- Los primeros sistemas de control de versionado, utilizaban una base de datos para llevar el registro de los cambios que se realizaban sobre los archivos



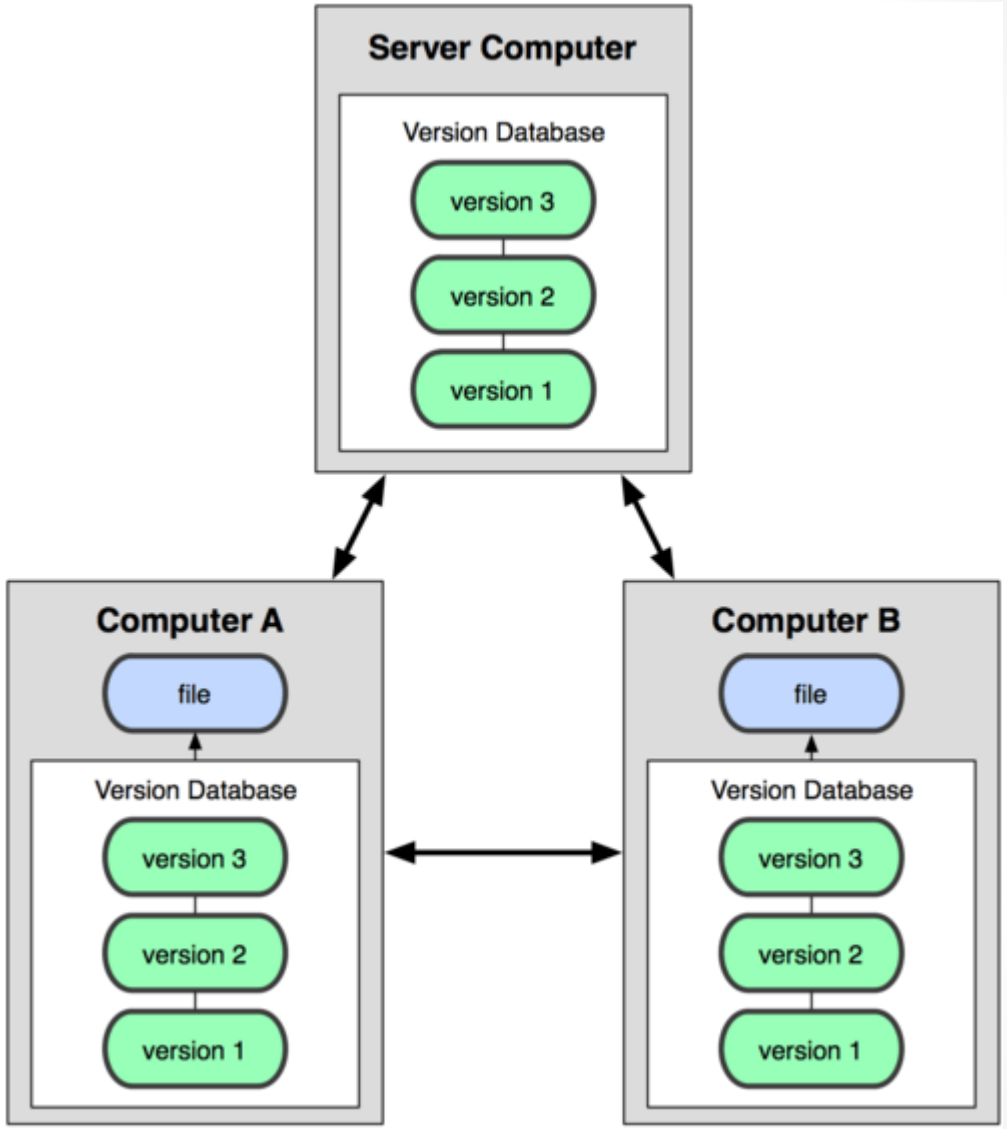
CVCS – Centralizados



- Con la necesidad de que otro desarrolladores puedan colaborar en el desarrollo de sistemas, surgieron los sistemas de control de versiones centralizados.
- Estos sistemas, como CVS, Subversion, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central.

DVCS - Distribuidos

- En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio.



Git – Historia

- Durante la mayor parte del mantenimiento del núcleo de Linux (1991-2002), los cambios en el software se pasaron en forma de parches y archivos. En 2002, el proyecto del núcleo de Linux empezó a usar un DVCS propietario llamado BitKeeper.
- En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y la herramienta dejó de ser ofrecida gratuitamente. Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta.

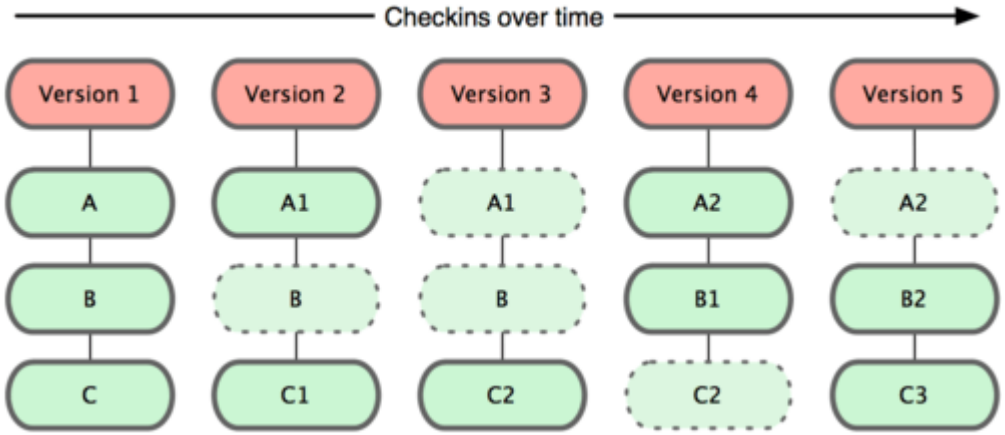
Git – Historia

Los objetivos del nuevo sistema fueron los siguientes:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Caracteristicas

- Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, básicamente se hace una foto del aspecto de todos los archivos en ese momento, y guarda una referencia a esa instantánea.



- Para ser eficiente, si los archivos no se han modificado, Git hace una referencia al archivo almacenado en la versión en la cual fue versionado.

Características - Integridad

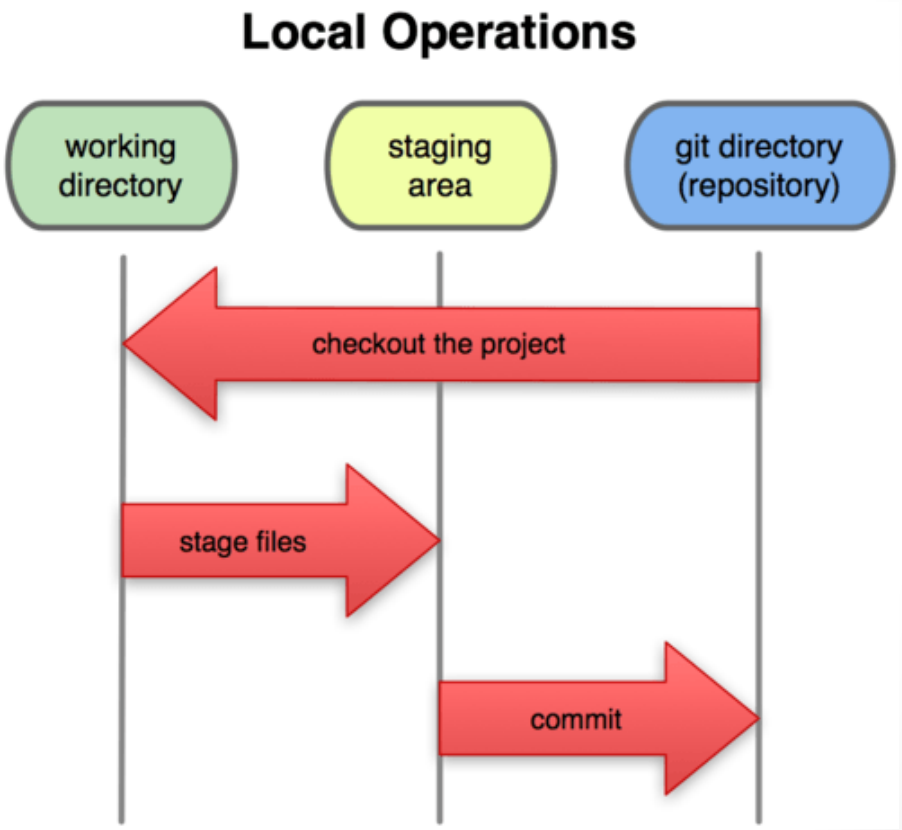
Todo en Git es verificado mediante una suma de comprobación (checksum en inglés) antes de ser almacenado, y es identificado a partir de ese momento mediante dicha suma. Esto significa que es imposible cambiar los contenidos de cualquier archivo o directorio sin que Git lo sepa.

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

Características - Estados

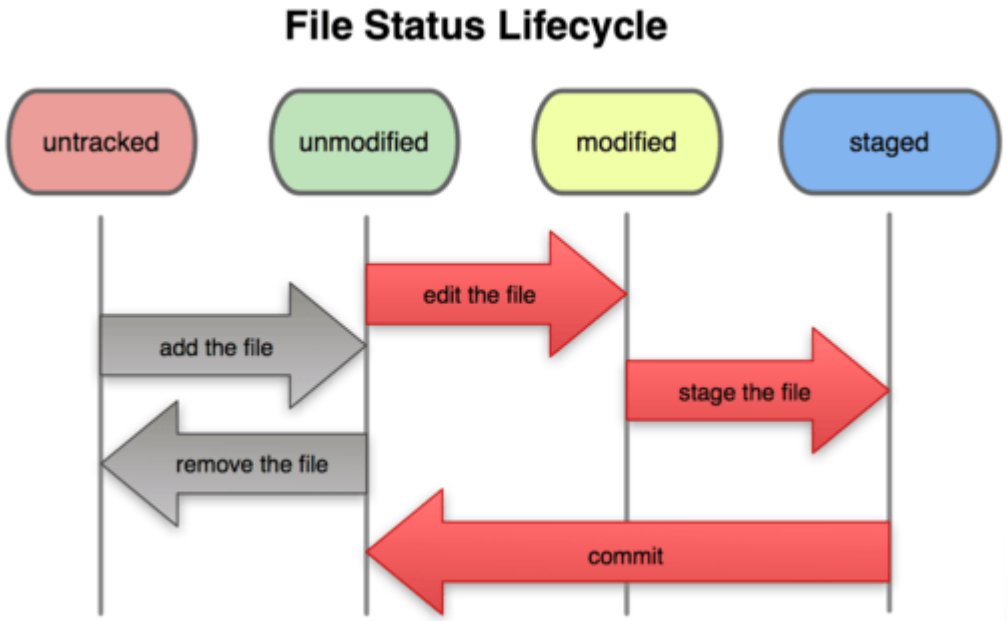
Git tiene tres estados principales en los que se pueden encontrar tus archivos:

- confirmado (committed): significa que los datos están almacenados de manera segura en tu base de datos local.
- modificado (modified): significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- preparado (staged). significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.



Estados de los archivos.

- Cada archivo en el directorio de trabajo puede estar: bajo seguimiento (tracked), o sin seguimiento (untracked). Los archivos bajo seguimiento son aquellos que existían en la última instantánea; pueden estar sin modificaciones, modificados, o preparados. Los archivos sin seguimiento son todos los demás.



Características - Estados

- El directorio .Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

- El directorio de trabajo es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que los puedas usar o modificar.

- El área de preparación es un sencillo archivo, generalmente contenido en tu directorio .Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.

Instalando en Windows

- El proyecto msysGit permite tener la versión de línea de comandos (con cliente SSH) y una interfaz grafica para usuario.

  http://msysgit.github.com/

- Los primeros pasos al instalar Git, es configurar un nombre y un correo para establecer una identidad que usará al momento de ralizar las confirmaciones de cambios (commits).

```
$ git config --global user.name "John Doe“
$ git config --global user.email johndoe@example.com
```

Formas de iniciar un Repositorio.

- Para iniciar un repositorio en un proyecto existente podemos hacer:

```
$ git init
```

(Esto crea un nuevo subdirectorio llamado .git que contiene todos los archivos necesarios del repositorio — un esqueleto de un repositorio Git).

- Para obtener una copia de un repositorio Git existente —por ejemplo, un proyecto en el que te gustaría contribuir— el comando que necesitas es

```
$ git clone 'direccion del repositorio origen' {opcionalmente carpeta destino}
```

Agregando nuevos archivos.

- Para empezar el seguimiento de un nuevo archivo se usa el comando git add. Iniciaremos el seguimiento del archivo README ejecutando esto:

```
$ git add README
```

- Si ejecutamos el comando git status, veremos que el README está ahora bajo seguimiento y preparado:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
##   new file:   README#
```

- Para ignorar archivos o carpetas, podemos crear una archivo .gitignore e incluirle los elementos que queremos excluir.

Confirmando cambios.

- Ahora que agregamos los archivos modificados al área de preparación, podemos confirmar los cambios. (solo se incluirán los archivos a los que le hayamos realizado el git add).
- La forma más fácil de confirmar es escribiendo git commit, pero es posible pasarle un mensaje con el parametro –m.

```
$ git commit -m "Story 182: Fix benchmarks for speed“

[master]: created 463dc4f: "Fix benchmarks for speed"
2 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 README
```

- Si nos falto realizar un cambio, agregar un archivo o bien queremos modificar el mensaje podemos modificar la ultima instantanea ejecutando un commit con la opción --amend:

```
$ git commit --amend
```

Trabajar con repositorios remotos.

- Los repositorios remotos son versiones de tu proyecto que se encuentran alojados en Internet o en algún punto de la red. Pueden ser de sólo lectura, o de lectura/escritura, según los permisos que tengas.

- Colaborar con otros implica gestionar estos repositorios remotos, y mandar (push) y recibir (pull) datos de ellos cuando necesites compartir cosas.

- Gestionar repositorios remotos implica conocer cómo añadir repositorios nuevos, eliminar aquellos que ya no son válidos, gestionar ramas remotas e indicar si están bajo seguimiento o no, y más cosas.

Consultar repositorios remotos y subir cambios.

- Para ver qué repositorios remotos tenemos configurados, podemos ejecutar el comando git remote.

```
$ git remote –v
origin  git://github.com/schacon/ticgit.git (fetch)
origin  git://github.com/schacon/ticgit.git (push)
```

- Para obtener las ultimas versiones del repositorio remoto hay 2 opciones:
- git fetch origin recupera toda la información servidor desde que lo clonaste (o desde la última vez que ejecutaste fetch). El comando fetch sólo recupera la información y la pone en tu repositorio local —no la une automáticamente con tu trabajo ni modifica aquello en lo que estás trabajando. Tendrás que unir ambos manualmente a posteriori.
- git pull para recuperar y unir automáticamente la rama remota con tu rama actual. Éste puede resultarte un flujo de trabajo más sencillo y más cómodo;

Subir cambios.

- Para enviar tus cambios al repositorio remoto.
- El comando que te permite subir los cambios es: git push [nombre-remoto][nombre-rama]. Si quieres enviar tu rama maestra (master) a tu servidor origen (origin), ejecutarías esto para enviar tu trabajo al servidor:

```
$ git push origin master
```

Sitios de interés y mas información:

- https://git-scm.com/ - Información sobre git
- https://github.com/ - Tu repositorio y herramientas en la nube.
- https://c9.io/ - Repositorios y herramientas de desarrollo en la nube.