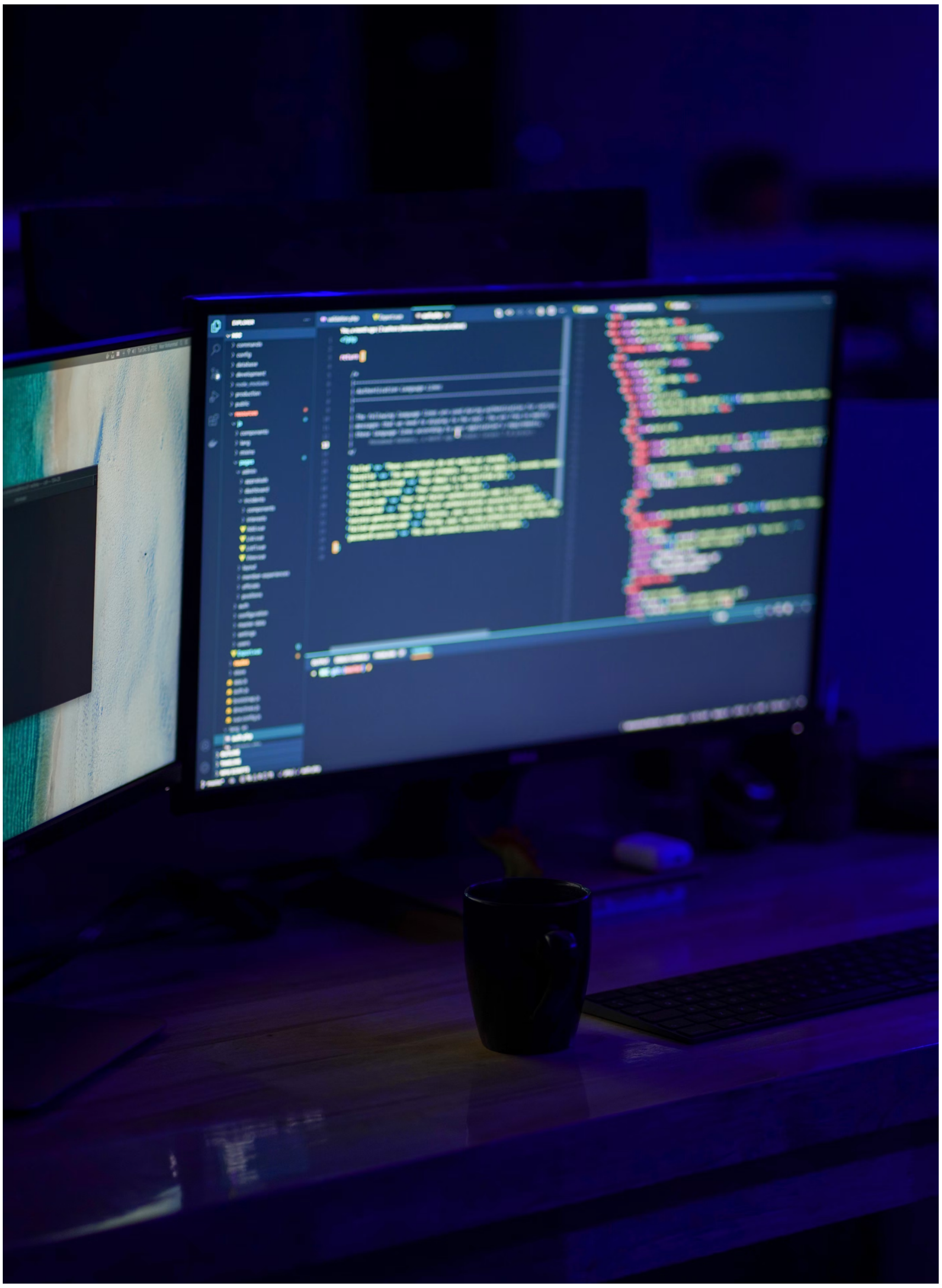
Lección 1:
GIT



Tiempo de ejecución: 4 horas

Planteamiento de la sesión

Materiales

- Git - Acerca del Control de Versiones
- Git - Downloads
- GIT CHEAT SHEET
- git - la guía sencilla
- Git - gittutorial Documentation
- 3.1 Ramificaciones en Git - ¿Qué es una rama?
- The Linux command line for beginners | Ubuntu

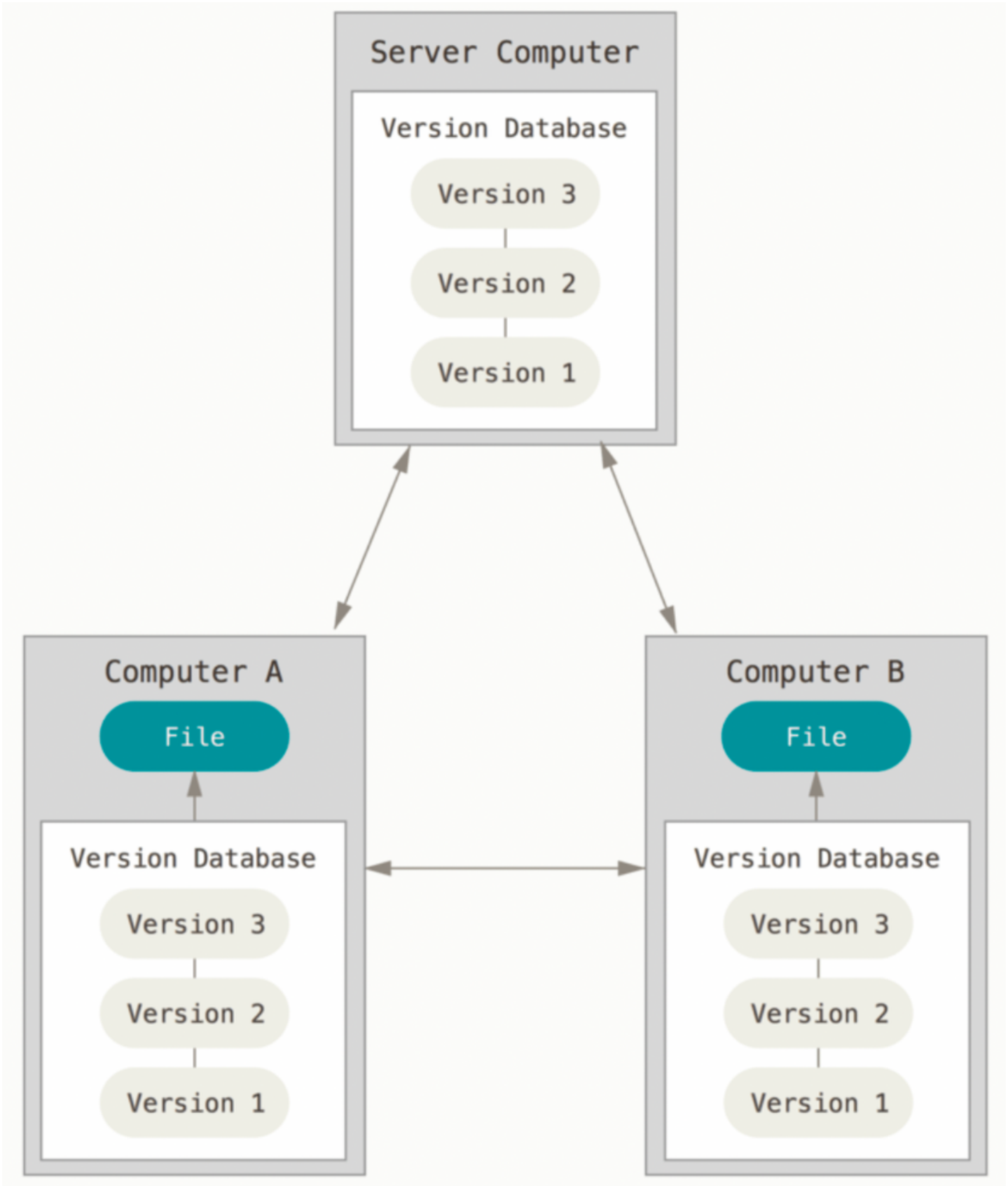
GIT - Control de Versiones

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Si eres diseñador gráfico o de web y quieres mantener cada versión de una imagen o diseño (es algo que sin duda vas a querer), usar un sistema de control de versiones (VCS por sus siglas en inglés) es una decisión muy acertada. Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente.

Existen controles de versiones locales y otros centralizados. El primero hace referencia a versionamiento dentro de una máquina de uno de los colaboradores en el proyecto, mientras que la segunda indica que se tiene un control de versiones central para todos los colaboradores en el proyecto, lo que permite a todo el equipo estar actualizado durante el proceso.

Una tercera visión, son los sistemas de control de versiones distribuidos, en los cuales cada colaborador tiene control de versiones local y adicional existe un control de versiones común para todos. Así cada colaborador tendrá completo control sobre su trabajo, pero también puede estar actualizado a la hora de trabajar en equipo.

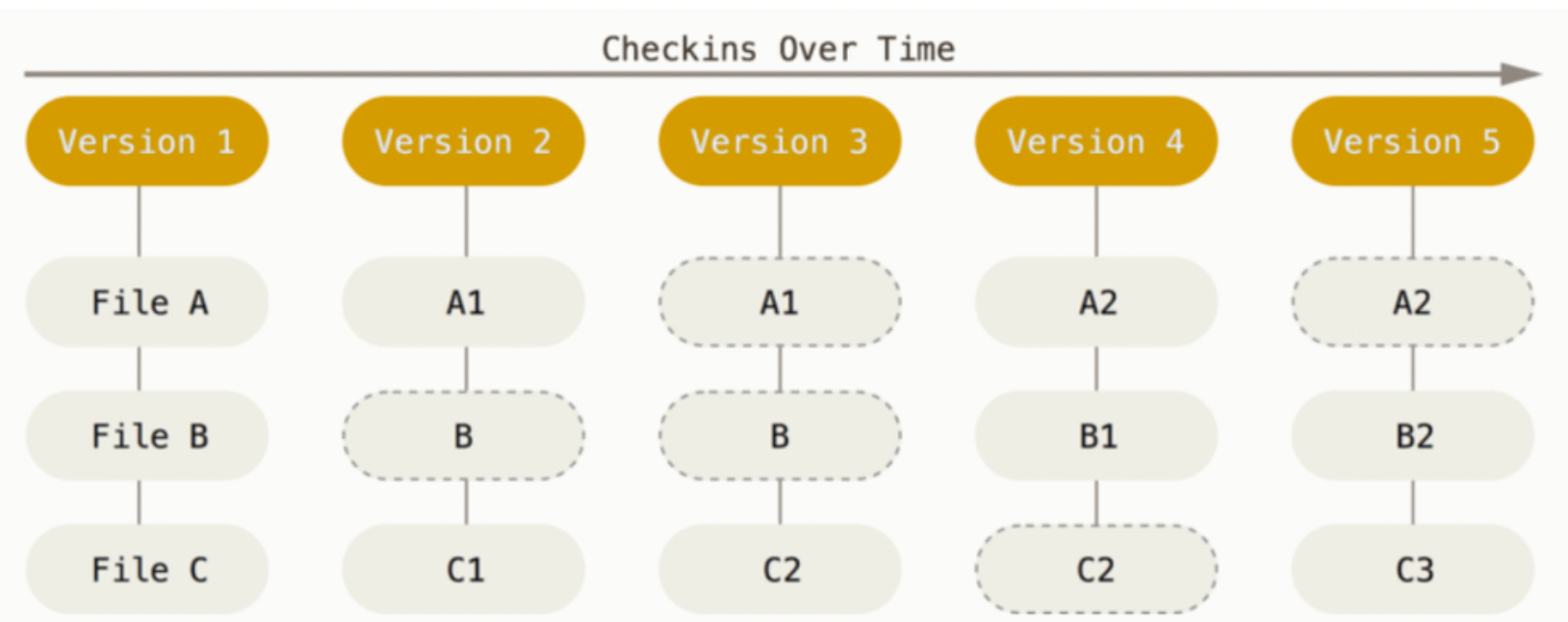


Fuente: Control de versiones distribuido. Consultado en marzo de 2024, disponible en internet, Git - Acerca del Control de Versiones

Una solución para todo el trabajo del versionamiento es GIT, que nació como una propuesta de la comunidad de desarrollo de Linux, encabezada por su creador, Linus Torvalds, para sustituir su anterior sistema de control de versiones que pasaba a ser de pago. Las lecciones aprendidas y lo que buscaban con GIT es que cuente con:

- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el kernel de Linux) eficientemente (velocidad y tamaño de los datos)

Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente toma una foto del aspecto de todos tus archivos en ese momento y guarda una referencia a esa copia instantánea. Para ser eficiente, si los archivos no se han modificado Git no almacena el archivo de nuevo, sino un enlace al archivo anterior idéntico que ya tiene almacenado. Git maneja sus datos como una secuencia de copias instantáneas.



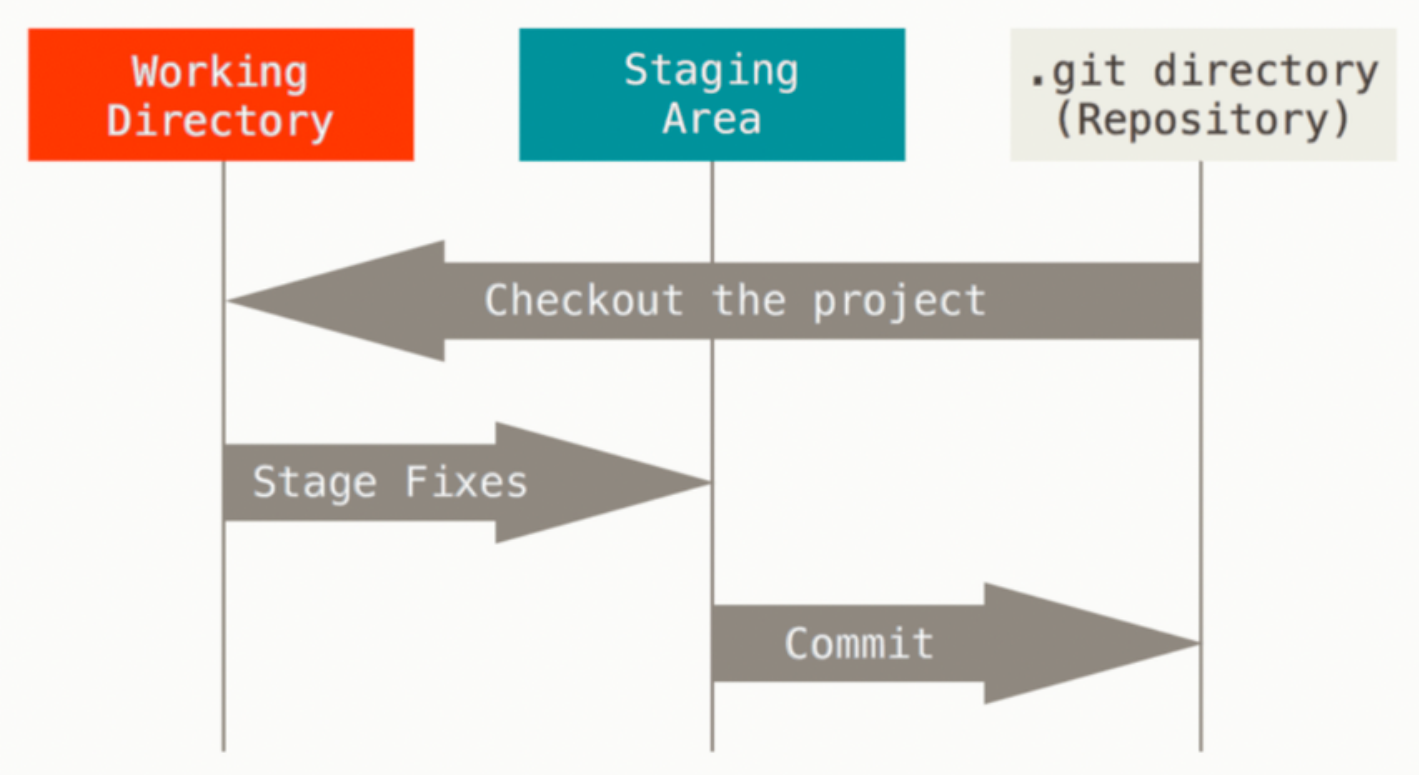
Fuente: Almacenamiento de datos como instantáneas del proyecto a través del tiempo. Consultado en marzo de 2024, disponible en internet, Fundamentos de Git

Esto hace que Git se parezca más a un sistema de archivos miniatura con algunas herramientas extremadamente poderosas desarrolladas sobre él, que a un VCS.

Los tres estados

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged). Confirmado: significa que los datos están almacenados de manera segura en tu base de datos local. Modificado: significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos. Preparado: significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: El directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).



Fuente: Directorio de trabajo, área de almacenamiento y el directorio Git. Consultado en marzo de 2024, disponible en internet, Fundamentos de Git

El directorio de Git es donde se almacenan 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 otra computadora.

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 archivo, generalmente contenido en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación. A veces se le denomina índice (“index”), pero se está convirtiendo en estándar el referirse a ella como el área de preparación.

El flujo de trabajo básico en Git es algo así:

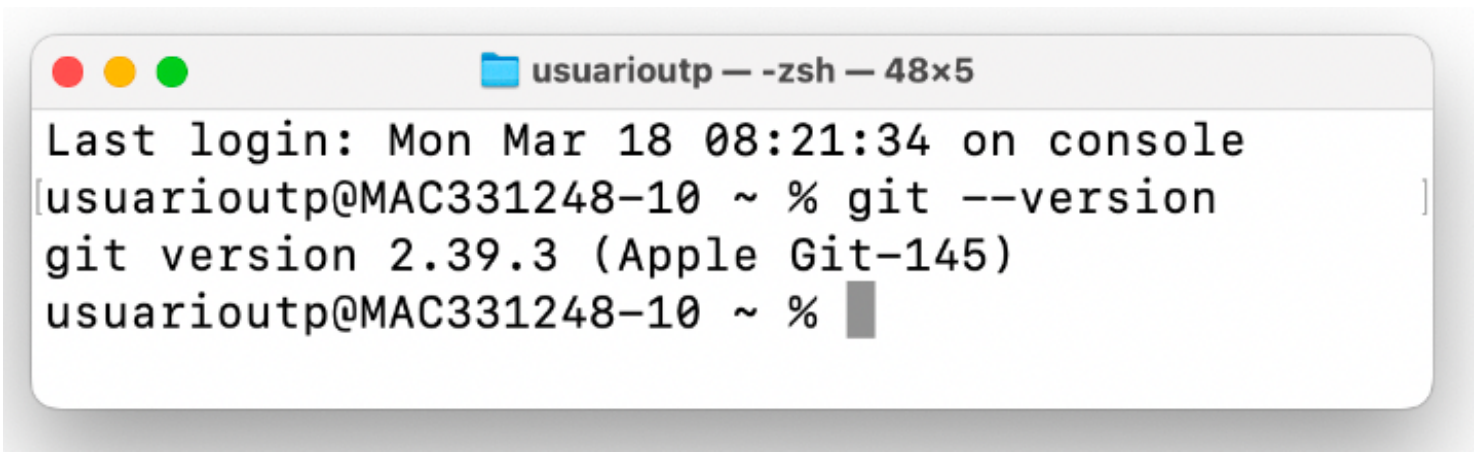
1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.

Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada (committed). Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada (staged). Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada (modified).

Instalación

La instalación es un proceso sencillo y depende del sistema operativo, la documentación oficial tiene guías para cada uno de los casos. Git - Downloads

- Ir a la web oficial y descargar el ejecutable (en windows o ejecutar el comando indicado en los otros S.O).
- Ejecutar el archivo que descargamos.
- Si tu sistema operativo es Windows, además de instalarse Git, se instalará en tu máquina una terminal llamada Git Bash, que permite ejecutar comando de unix dentro de Windows. The Linux command line for beginners | Ubuntu
- Una vez instalado Git, estará disponible el comando git para correr en la terminal.
- Para verificar que la instalación se haya realizado correctamente, abrir una terminal y correr el comando git --version.



Debería finalizar con algo como lo presentado en la imagen anterior.

Configurando

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 "John Doe"
git config --global user.email johndoe@example.com

#Y verificarlos con

git config user.name
git config user.email

También es posible modificar el editor por defecto de Git, sin embargo, si se hace desde windows, el proceso de instalación tiene una opción donde se puede seleccionar de una lista disponible.

Iniciando un repo

Si estás empezando a seguir un proyecto existente en Git, debes ir al directorio del proyecto y usar el siguiente comando:

$ git init

Esto crea un subdirectorio nuevo llamado .git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.

```
drwxr-xr-x@  5 usuarioutp  staff    160 Jun 15  2023 .
drwxr-xr-x@ 13 usuarioutp  staff    416 Jan 15 09:53 ..
drwxr-xr-x  14 usuarioutp  staff    448 Mar 13 16:02 .git
drwxr-xr-x   6 usuarioutp  staff    192 Jun  5  2023 assets
-rw-r--r--   1 usuarioutp  staff  28911 Mar 13 16:01 index.html
```

La carpeta es oculta por lo que necesitas habilitar los elementos ocultos en el explorador de archivos en Windows o el comando

ls -al

En Mac o Linux.

Si deseas empezar a controlar versiones de archivos existentes (a diferencia de un directorio vacío), probablemente deberías comenzar el seguimiento de esos archivos y hacer una confirmación inicial. Puedes conseguirlo con unos pocos comandos git add para especificar qué archivos quieres controlar, seguidos de un git commit para confirmar los cambios:

```
#todos los archivos con extensión .c
$ git add *.c

#el archivo llamado LICENSE
$ git add LICENSE

#confirmar los cambios con el mensaje 'initial project version'
$ git commit -m 'initial project version'
```

También es posible no iniciar de un proyecto propio sino clonar un repositorio remoto y trabajar a partir de allí, por ejemplo:

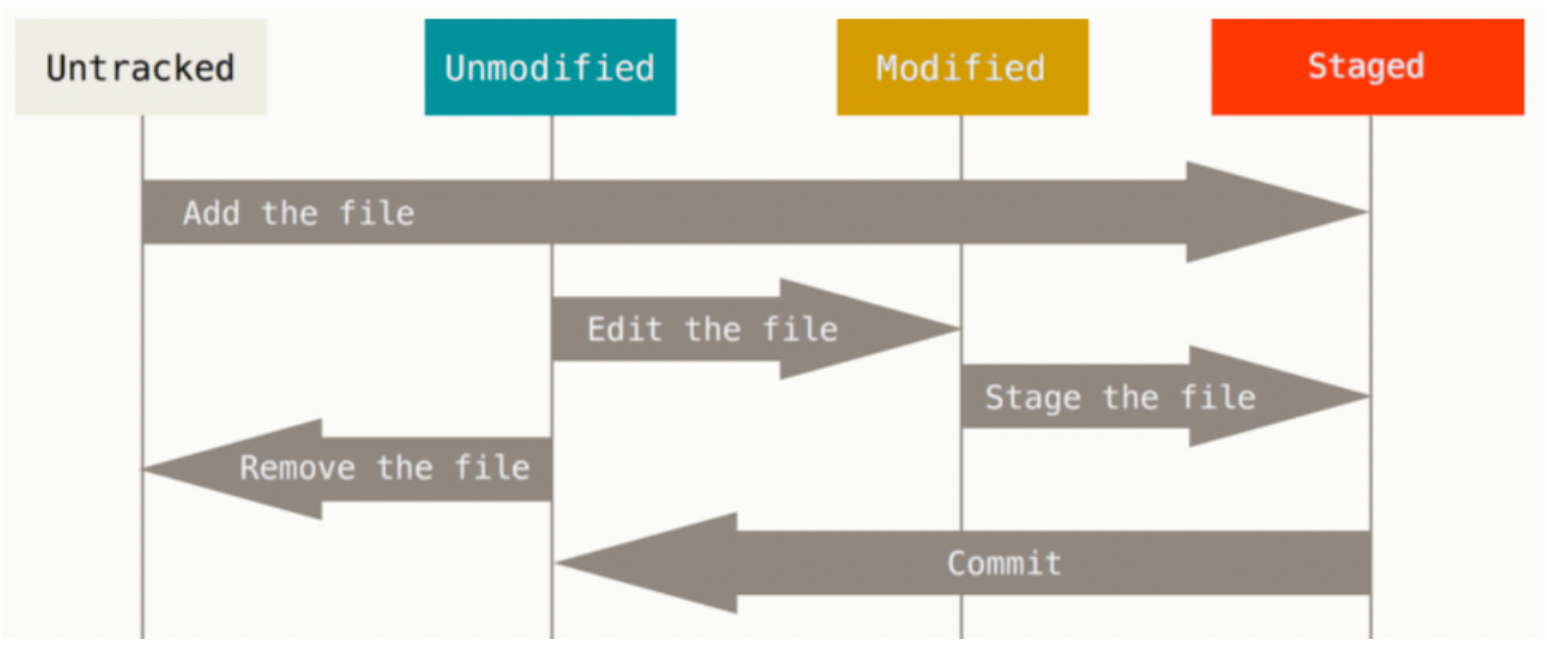
```
$ git clone https://github.com/libgit2/libgit2
```

Guardando cambios

Cada archivo de tu repositorio puede tener dos estados: rastreados y sin rastrear. Los archivos rastreados (tracked files en inglés) son todos aquellos archivos que estaban en la última instantánea del proyecto; pueden ser archivos sin modificar, modificados o preparados. Los archivos sin rastrear son todos los demás - cualquier otro archivo en tu directorio de trabajo que no estaba en tu última instantánea y que no está en el área de preparación (staging area). Cuando clonas por primera vez un repositorio, todos tus archivos estarán rastreados y sin modificar pues acabas de sacarlos y aún no han sido editados.

Mientras editas archivos, Git los ve como modificados, pues han sido cambiados desde su último commit. Luego preparas estos archivos modificados y finalmente confirma todos los cambios preparados, y repites el ciclo.

Fuente: El ciclo de vida del estado de tus archivos, Consultado en marzo de 2024, disponible en internet, Git - Guardando cambios en el Repositorio



- DML – Lenguaje de manipulación de datos
- DCL – Lenguaje de control de datos
- TCL – Lenguaje de control de transacciones

Fuente: SQL Commands, consultado en febrero de 2024, disponible en SQL | DDL, DQL, DML, DCL and TCL Commands - GeeksforGeeks.

De los 5 grandes bloques se componen todo el lenguaje SQL, nos centraremos en 3 categorías

Comandos base

```
#Revisar el estado de los archivos en el work directory
git status

#Comenzar a rastrear nuevos archivos o agregar archivos modificados

git add .
git add file_name_1, file_name_2

#confirmar los cambios
git commit -m "Message Here"
```

Ignorando archivos

Git ofrece la manera de ignorar archivos, tipos de archivos o directorios completos, esto se hace creando un archivo llamado .ignore, si, con el punto al inicio.

Las reglas sobre los patrones que puedes incluir en el archivo .gitignore son las siguientes:

- Ignorar las líneas en blanco y aquellas que comiencen con #.
- Emplear patrones glob estándar que se aplicarán recursivamente a todo el directorio del repositorio local.
- Los patrones pueden comenzar en barra (/) para evitar recursividad.
- Los patrones pueden terminar en barra (/) para especificar un directorio.
- Los patrones pueden negarse si se añade al principio el signo de exclamación (!).

Los patrones glob son una especie de expresión regular simplificada usada por los terminales. Un asterisco (*) corresponde a cero o más caracteres; [abc] corresponde a cualquier caracter dentro de los corchetes (en este caso a, b o c); el signo de interrogación (?) corresponde a un caracter cualquiera; y los corchetes sobre caracteres separados por un guión ([0-9]) corresponde a cualquier caracter entre ellos (en este caso del 0 al 9). También puedes usar dos asteriscos para indicar directorios anidados; a/**/z coincide con a/z, a/b/z, a/b/c/z, etc.

Ejemplo:

```
# ignora los archivos terminados en .a
*.a

# pero no lib.a, aun cuando había ignorado los archivos terminados en .a en la línea anterior
!lib.a

# ignora unicamente el archivo TODO de la raiz, no subdir/TODO
/TODO

# ignora todos los archivos del directorio build/
build/

# ignora doc/notes.txt, pero no este: doc/server/arch.txt
doc/*.txt

# ignora todos los archivos .txt del directorio doc/
doc/**/*.txt
```

Eliminar archivos

Para eliminar archivos de Git, debes eliminarlos de tus archivos rastreados (o mejor dicho, eliminarlos del área de preparación) y luego confirmar. Para ello existe el comando git rm, que además elimina el archivo de tu directorio de trabajo de manera que no aparezca la próxima vez como un archivo no rastreado.

```
$ git rm PROJECTS.md

rm 'PROJECTS.md'

$ git status

On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

deleted:  PROJECTS.md
```

Con la próxima confirmación, el archivo habrá desaparecido y no volverá a ser rastreado.

Otra cosa que puedas querer hacer es mantener el archivo en tu directorio de trabajo pero eliminarlo del área de preparación. En otras palabras, quisieras mantener el archivo en tu disco duro pero sin que Git lo siga rastreando. Esto puede ser particularmente útil si olvidaste añadir algo en tu archivo .gitignore y lo preparaste accidentalmente, algo como un gran archivo de trazas a un montón de archivos compilados .a. Para hacerlo, utiliza la opción --cached:

```
$ git rm --cached README
```

Historial

Después de haber hecho varias confirmaciones, o si has clonado un repositorio que ya tenía un histórico de confirmaciones, probablemente quieras mirar atrás para ver qué modificaciones se han llevado a cabo. La herramienta más básica y potente para hacer esto es el comando git log.

```
$ git log

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

changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 10:31:28 2008 -0700

first commit

Deshacer cambios

Por ahora, deshacer cosas será guíada por las opciones que entrega Git al momento de un git status, ya sea desconfirmar un archivo confirmado o revertir un archivo a su versión anterior.

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

renamed: README.md -> README
modified: CONTRIBUTING.md

Git nos indica que si quiere quitar algún archivo de la confirmación debe realizar el comando git reset (mucho ojo aquí, solo utilizarlo como lo indica el mensaje)

$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M CONTRIBUTING.md

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

renamed: README.md -> README

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Algo similar ocurre al momento de revisar los archivos modificados

$ git status

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Al igual que con el comando reset, vamos a utilizar el checkout solo como lo indica el mensaje.

$ git checkout -- CONTRIBUTING.md

Finalmente, una de las acciones más comunes a deshacer es cuando confirmas un cambio antes de tiempo y olvidas agregar algún archivo, o te equivocas en el mensaje de confirmación. Si quieres rehacer la confirmación, puedes reconfirmar con la opción --amend:

Por ejemplo, si confirmas y luego te das cuenta que olvidaste preparar los cambios de un archivo que querías incluir en esta confirmación, puedes hacer lo siguiente:

$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend

Por ahora, existen las bases necesarias para trabajar con Git, hay un punto adicional que se trabaja más adelante y es el tema de los remotos, repositorios en la nube, como por ejemplo Github.

Otro punto como lectura sugerida, que es importante y hace parte de una de las fortalezas de Git es el branching o trabajo en ramas o ramificaciones en Git 3.1 Ramificaciones en Git - ¿Qué es una rama? junto con el git workflow.

Resumen

COMANDOS PASO A PASO PARA CREAR UN REPOSITORIO LOCAL

```
>_ git init  // crea el repositorio
```

```
>_ git config user.name "nombreUsuario" // agrega nuestra identidad
```

```
>_ git config user.email "emailUsuario" // agrega nuestro e-mail
```

```
>_ git remote add origin http://... // apunta al repositorio remoto
```

COMANDOS PASO A PASO PARA SUBIR CAMBIOS

```
>_ git add .  // agrega todos los archivos
```

```
>_ git commit -m "mensaje"  // commitea los cambios hechos
```

```
>_ git push origin master  // envía los cambios al repositorio remoto
```

```
>_ git status // realiza un seguimiento de los estados de los archivos
```

```
>_ git status // realiza un seguimiento de los estados de los archivos
```

```
>_ git pull// descarga los cambios que existen en el repositorio remoto
```

Aquí también puede encontrar una guía resumida de Git, lo que permite realizar consultas rápidas en caso de necesitar recordar algún concepto git - la guía sencilla

También está la hoja de trucos, donde puedes consultar un listado extenso de comandos y su uso. GIT CHEAT SHEET
index :: Git Cheatsheet :: NDP Software;

Git - gittutorial Documentation

Ejercitaciones Git

Oh My Git!

learn git branching

Learn Git with Bitbucket Cloud | Atlassian Git Tutorial

Git Tutorial

Learn Git & GitHub | Codecademy