1. What is Git?

1.1. Snapshots, Not differences

Los sistemas VCS usuales tienen un control de versiones *delta-based*. Guardan listas de cambios por archivo.

Git guarda snapshots de todos los archivos modificados y guarda apuntadores a cada estado. Cada repositorio de git es una copia completa y funcional del proyecto, no requiere dependencias externas y es todo local.

Git hace checksums en cada commit, lo cual hace difícil o imposible modificar sin que Git se de cuenta o perder información porque todo se verifica y valida.

1.2. The Three States

Git tiene tres estados en los que puede estar un archivo en cualquier momento dado:

- **Modified**: El archivo tiene modificaciones pero aún no se le ordena a Git incluir los cambios en el siguiente "commit" o confirmación.
- **Staged / Preparado**: El archivo fue modificado y marcado para que Git lo observe y guarde su versión actual en el siguiente commit.
- **Commited / Confirmado**: Los cambios al archivo ya fueron observados y confirmados. Se guarda el estado actual del archivo en la base de datos de Git.

Adicionalmente hay archivos que pueden estar dentro del directorio, en disco, y no existir dentro de la base de datos Git. Los tres estados anteriores solo aplican para archivos *rastreados* o *tracked*. Los archivos *no-rastreados* o *untracked* siguen dentro del alcance de búsqueda de Git, y éste aún los puede ver, pero tiene instrucciones de no rastrear modificiaciones.

Similarmente, hay tres secciones o "espacios" en cada proyecto de Git:

1. Working Directory.
2. Staging Area.
3. Git Repository.

Working Directory es el directorio "fisico", local en tu máquina en donde estás trabajando. Es como cualquier otra carpeta, pero tiene la particularidad de que hay una instancia de Git observándola. El *Working Directory* es el checkout de una versión particular del proyecto. Los archivos fueron descomprimidos de la base de datos de Git, y puestos en el disco para poder ser modificados como cualquier otro archivo.

Staging Area se puede pensar como el lugar físico a donde se mandan los archivos que están marcados como listos para ser confirmados (commited). En realidad es un archivo, pero la abstracción de lugar es más útil.

Git repository es donde vive la base de datos de Git, y todos los metadatos asociados. Aqui se guardan las diferentes versiones comprimidas de cada commit hecho, asi como apuntadores y metadatos de documentación. El archivo .git es el que contiene toda esta información, y es el que obtiene uno al clonar un repositorio.

Un archivo particular se considera *commited* (confirmado) si fue modificado, añadido al *staging area*, y se hizo un *commit* (confirmación). Los cambios que se le hicieron a ese archivo ahora están grabados en git. Un archivo se considera *staged* (preparado) si se modificó y se añadió al *staging area*. Un archivo se considera *modified* (modificado) si sufrió cambios desde la última versión que conoce Git, pero aún no se añade al *staging area*. Los archivos *modified* que no sean añadidos al *staging area* antes de efectuar un *commit* no serán rastreados en ese *commit*, y sus cambios no se guardarán.

1.3. Basic Git Workflow

1. Modifica archivos locales en disco.
2. Se elige qué archivos se desea rastrear (*track*) añadiéndolos al *staging area*. Son estos y solo estos los archivos que serán parte de la siguente confirmación.
3. Se lleva a cabo una *confirmación*, en la que se toman un snapshot del estado actual del *staging area* y se guarda permanentemente en el repositorio Git, junto con datos de identificación y mensajes de confirmación.

1.4. Command Line Interface

Para obtener ayuda sobre el comando

```
$ git add -h
usage: git add [<options>] [--] <pathspec>...

    -n, --dry-run         dry run
    -v, --verbose         be verbose

    -i, --interactive     interactive picking
    -p, --patch           select hunks interactively
    -e, --edit            edit current diff and apply
    -f, --force           allow adding otherwise ignored
       files
    -u, --update          update tracked files
    --renormalize         renormalize EOL of tracked files (
       implies -u)
    -N, --intent-to-add   record only the fact that the path
       will be added later
    -A, --all             add changes from all tracked and
       untracked files
    --ignore-removal      ignore paths removed in the working
        tree (same as --no-all)
    --refresh             don't add, only refresh the index
```

```
    --ignore-errors       just skip files which cannot be
       added because of errors
    --ignore-missing      check if - even missing - files are
        ignored in dry run
```

2. Git Basics

2.1. Getting a Git Repository

Usualmente uno obtiene un repositorio de Git en dos maneras:

1. Tomas cualquier carpeta local en tu disco, y la conviertes en un repositorio con `git init`, o bien
2. *Clonas* un repositorio de Git existente de algún otro lugar.

Para crear un repositorio nuevo desde una carpeta local, basta con navegar hasta la carpeta deseada a través de la terminal de comandos, y ejecutar el comando `git init` que creará un repositorio nuevo con todo lo necesario para usar Git, y rastrear cambios en los archivos de ese directorio. El comando `init` crea un "esqueleto de git". En este punto, ningun archivo está siendo rastreado, todos aparecen como *untracked*. Si deseas empezar a rastrear cambios, debes cambiar el estatus de los archivos de *untracked* a *tracked* a través del commando `git add`. Cabe mencionar que los archivos nuevos, es decir los originales en un directorio antes de correr `git init`, o aquellos creados o añadidos después de haber creado el repositorio, siempre aparecerán por primera vez como *untracked*, hasta que se le señale a Git que se deben rastrear.

> Notas: El comando `git add` tiene dos funciones: Cambia el estatus de un archivo de *untracked* a *tracked*, y además añade archivos *modified* al *staging area* para prepararlos para un *commit*.
>
> `add` recibe como argumento nombres de archivos, o patrones *glob*.

Tradicionalmente una vez que se han rastreado los archivos de interés se hace un "primer commit".

```
$ git commit -m "Primer commit"
```

El modificador `-m` es corto para `message`, y quiere decir que le mensaje de confirmación viene en seguida rodeado de comillas, como en el ejemplo de arriba. Si no se usa el modificador (flag) `-m`, Git abrirá el editor de texto default de la terminal, por ejemplo Vim, y estarás atrapado y confundido.

Si deseas copiar un repositorio existente, por ejemplo, para contribuir a algún proyecto o con tu equipo, entonces se crea un nuevo repositorio con `git clone`. El comando `clone` recibe de argumento un URL a algún repositorio remoto. Por ejemplo, el comando a continuación clona el repositorio donde está alojado el libro Pro Git, de Scott Chacon, con base en el cual se hicieron estas notas.

```
$ git clone https://github.com/progit/progit2
```

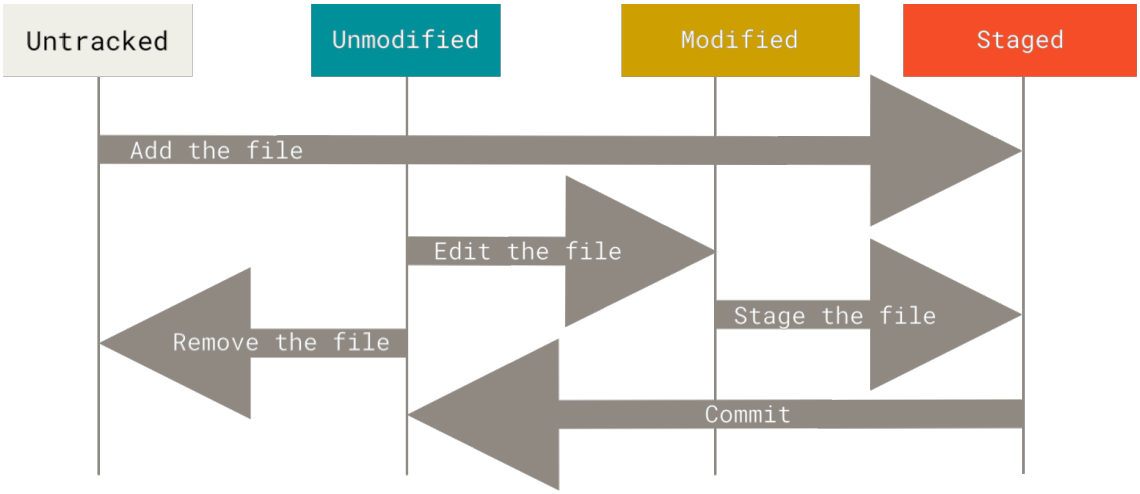

Figura 1: Ciclo de vida de un archivo en un Git Repo.

El comando clone crea una nueva carpeta en el directorio actual donde residirá el repositorio `.git` junto con los archivos en el que corresponde a la versión que clonaste. En automático el nombre de la carpeta que se va a crear es el nombre del repositorio, en este caso "progit2", pero también es posible especificar un nombre distinto. Por ejemplo, si queremos que la nueva carpeta se llame "libro-git", pasamos ese nombre como argumento adicional, es decir:

```
$ git clone https://github.com/progit/progit2 libro-git
```

2.2. Recording Changes

Una vez que se tiene un repositorio de Git y archivos rastreados, se puede empezar a usar todo el potencial de Git.

Como habíamos dicho, un archivo puede estar en uno de cuatro estados en cualquier momento dado: *untracked*, *unmodified*, *modified*, *staged*, como lo muestra la figura 1.

Para checar en qué punto del ciclo se encuentran los archivos existe el comando `git status`. Por ejemplo, al correrlo cuando se ha iniciado un nuevo repo o no se han hecho cambios desde el último *commit*, se ve asi:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Con la frase "working directory clean" quiere decir que no hay cambios para el siguiente *commit*, ya sea porque no se han rastreado los archivos o porque los archivos rastreados no han sido modificados desde el último *commit*. Si se añaden archivos nuevos desde el último commit al correr `git status` saldrá un mensaje similar:

```
Untracked files:
```

```
  (use "git add <file>..." to include in what will be
     committed)

        ejemplo.txt
```

con lo cual vemos que el archivo es reconocido por Git, pero que los cambios que se le hagan o su estado actual no serán rastreados por git.

Una vez que se empieza a rastrear un archivo y se hace un primer *commit* con el y se empiece a modificar, la siguiente vez que se corra `git status`, mostrará algo como lo siguiente:

```
On branch master
  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:   ejemplo.txt
```

. Eso no quiere decir que el archivo (en este caso `ejemplo.txt`) dejó de ser rastreado por Git y que se perdió el historial de cambios, sino que Git reconoce los cambios hechos al archivo y ahora está esperando a que se añada al *staging area* mediante `git add`, ya que Git deja al usuario decidir qué cambios se toman en cuenta para un commit dado, en vez de asumir que todos los cambios entre un *commit* y el siguiente son todos relacionados con lo mismo.