Programación Avanzada
Daniel Gregorio Longino
Tarea 1

# GitHub

## ¿Qué es Git?

Git es un sistema de control de versiones. Un sistema de control de versiones es una pieza de software diseñada para realizar un seguimiento de los cambios realizados en los archivos a lo largo del tiempo. Más específicamente, Git es un sistema de control de versiones distribuido, lo que significa que todos los que trabajan con un proyecto en Git tienen una copia del historial completo del proyecto, no solo el estado actual de los archivos.

## ¿Qué es GitHub?

GitHub es un sitio web en donde puede cargar una copia de su repositorio Git. Le permite colaborar mucho más fácilmente con otras personas en un proyecto. Lo hace al proporcionar una ubicación centralizada para compartir el repositorio, una interfaz basada en web para verlo y funciones como forking, pull requests, issues, y wikis, que le permiten especificar, discutir y revisar los cambios con su equipo más efectivamente.

## ¿Por qué usar Git?

Incluso si está trabajando por su cuenta, si está editando archivos de texto, hay una serie de beneficios al usar Git. Esos beneficios incluyen lo siguiente:

1. *La capacidad de deshacer los cambios.* Si comete un error, puede volver a un momento anterior para recuperar una versión anterior de su trabajo.

2. *Una historia completa de todos los cambios.* Si alguna vez desea ver cómo se veía su proyecto hace un día, una semana, un mes o un año, puede consultar una versión anterior del proyecto para ver exactamente cuál era el estado de los archivos en ese momento.

3. *Documentación de por qué se realizaron los cambios.* A menudo es difícil recordar por qué se hizo un cambio. Con los mensajes de confirmación en Git, es fácil documentar para referencia futura del por qué está haciendo un cambio.

4. *La confianza para cambiar cualquier cosa.* Debido a que es fácil recuperar una versión anterior de su proyecto, puede tener la confianza de realizar los cambios que desee. Si no funcionan, siempre puede volver a una versión anterior de su trabajo.

5. *Múltiples flujos de historia.* Puede crear diferentes ramas de la historia para experimentar con diferentes cambios en su contenido o para desarrollar diferentes funciones de forma independiente. Luego, puede fusionarlos nuevamente en el historial del proyecto principal (the master branch) una vez que hayan terminado, o eliminarlos si no funcionan. Al trabajar en equipo, obtiene una gama aún más amplia de beneficios cuando usa Git para realizar un seguimiento de sus cambios. Algunos de los beneficios clave de Git cuando se trabaja con un equipo son:

6. *La capacidad de resolver conflictos.* Con Git, varias personas pueden trabajar en el mismo archivo al mismo tiempo. Por lo general, Git podrá fusionar los cambios automáticamente. Si no puede, le mostrará cuáles son los conflictos y le facilitará resolverlos.

7. *Flujos independientes de la historia.* Diferentes personas en el proyecto pueden trabajar en diferentes ramas, lo que le permite trabajar en características separadas de forma independiente y luego fusionar las características cuando están hechas.

## ¿Por qué usar GitHub?

GitHub es mucho más que un lugar para almacenar sus repositorios de Git. Proporciona una serie de beneficios adicionales, incluida la capacidad de hacer lo siguiente:

1. *Requisitos del documento.* Usando *Issues*, puede documentar errores o especificar nuevas funciones que le gustaría que su equipo desarrollara.

2. *Colabore en flujos independientes de la historia.* Usando ramas y *pull requests*, puede colaborar en diferentes ramas o características.

3. *Revisar el trabajo en curso.* Al mirar una lista de *pull requests*, puede ver todas las diferentes características en las que se está trabajando actualmente, y al hacer clic en cualquier *pull requests*, puede ver los últimos cambios, así como todas las discusiones sobre los cambios.

4. *Ver el progreso del equipo.* Revisar el *commit history* le permite ver en qué ha estado trabajando el equipo.

## El lugar de Git en GitHub

GitHub es un host para repositorios Git. En algún momento, es útil colocar su repositorio de Git en una ubicación compartida como respaldo y como un lugar donde otros pueden colaborar con usted en su código. Como host de Git, GitHub proporciona todas las funciones de Git además de algunos servicios útiles adicionales.

En GitHub, los proyectos o repositorios se almacenan en servidores remotos de GitHub. Si guarda todo su código en GitHub y su computadora falla, aún puede acceder a el. Aquí hay una lista de algunas características principales de Git que admite GitHub:

1. *Repositorio*: Cada repositorio contiene todos los archivos y carpetas relacionados con tu proyecto y te da el control de los permisos y la interacción de los colaboradores con tu código.

2. *Clone*: Cuando desee realizar cambios en su código, a menudo querrá crear una copia o clonación del proyecto en su computadora local. El proyecto clonado todavía está estrechamente conectado con la versión en GitHub; es solo su copia local.

3. *Fork*: Bifurcar un proyecto es cuando creas tu propia copia de todo el proyecto. Cuando bifurca un proyecto, GitHub crea un nuevo repositorio con su copia de todos los archivos. Todavía puede sugerir cambios a la copia original, pero también puede tomar su versión e ir en una nueva dirección.

4. *Branches*: GitHub admite la bifurcación e incluso proporciona una herramienta útil (pull request) para comparar la diferencia entre ramas y fusionar ramas.

5. *Commits*: GitHub realiza un seguimiento de todas los commits que envía a sus servidores y le brinda una interfaz sencilla para explorar el código en diferentes ramas y commits.

## Conceptos clave

Hay una serie de conceptos clave que deberá comprender para trabajar de manera efectiva con Git y GitHub.

1. *Commit*. Cada vez que guarda sus cambios en uno o más archivos en el historial en Git, crea un nuevo commit.

2. *Commit message*. Cada vez que realiza un commit, debe proporcionar un mensaje que describa por qué se realizó el cambio. Ese mensaje Commit message es invaluable cuando se intenta comprender más tarde por qué se implementó un determinado cambio.

3. *Branch*. Una serie independiente de commits que puede usar para probar un experimento o crear una nueva función.

4. *Master branch (master)*. Cada vez que crea un nuevo proyecto de Git, se crea una rama predeterminada que se llama Master branch. Esta es la rama en la que su trabajo debería terminar eventualmente una vez que esté listo para pasar a producción.

5. *Feature branch*. Siempre que esté creando una nueva funcionalidad, creará una rama para trabajar en ella. Eso se llama una feature branch.

6. *Release branch*. Si tiene un proceso de control de calidad manual o tiene que admitir versiones antiguas de su software para sus clientes, es posible que necesite una rama de lanzamiento como lugar para realizar las correcciones o actualizaciones necesarias.

7. *Merge*. Esta es una forma de tomar el trabajo completado de una rama e incorporarlo a otra rama. Por lo general, fusionará una feature branch en la rama principal.

8. *Tag*. Una referencia a un commit específico. Se utiliza con mayor frecuencia para documentar versiones de producción para que se sepa exactamente qué versiones del código entraron en producción y cuándo.

9. *Check out*. Ir a una versión diferente del historial del proyecto para ver los archivos a partir de ese momento. Por lo general, consultará una rama para ver todo el trabajo que se ha realizado en ella, pero se puede verificar cualquier commit.

10. *Pull request*. El Pull Request es el núcleo del sistema colaborativo de GitHub. Cuando haces un pull request estás en realidad proponiendo cambios para que alguien lo integre dentro del proyecto. GitHub te permite comparar el contenido de las dos ramas (una con tu propuesta y otra sin ella) de modo que con un sencillo código de colores puedas ver cuáles son las diferencias. Los cambios, adiciones y eliminaciones, se muestran en verde o rojo y se llaman diffs (diferencias).

11. *Issue*. Un Issue es una nota en un repositorio que trata de llamar la atención sobre un problema. Puede ser un error a corregir, una petición para añadir una nueva opción o característica, una pregunta para aclarar algún tema que no está correctamente aclarado o muchas otras cosas diferentes. En GitHub puedes etiquetar, buscar o asignar Issues, haciendo que la gestión de un proyecto activo sea más sencilla.

12. *Wiki*. Originalmente desarrollado por Ward Cunningham, los wikis son una forma ligera de crear páginas web con enlaces simples entre ellas. Los proyectos de GitHub a menudo usan wikis para la documentación.

13. *Clone*. A menudo querrá descargar una copia de un proyecto de GitHub para poder trabajar en él localmente. El proceso de copiar el repositorio a su computadora se llama clonación.

14. *Fork*. A veces, no tiene el permiso necesario para realizar cambios directamente en un proyecto. Tal vez sea un proyecto de código abierto escrito por personas que no conoce o es un proyecto escrito por otro grupo en su empresa con el que no trabaja mucho. Si desea enviar cambios a dicho proyecto, primero debe hacer una copia del proyecto en su cuenta de usuario en GitHub. Ese proceso se llama bifurcar el repositorio. Luego puede clonarlo, realizar cambios y enviarlos nuevamente al proyecto original mediante una solicitud de incorporación de cambios.