**Ideas clave: Introducción e instalación**

* Este primer módulo ha servido para familiarizarnos con Git y GitHub, dos programas de control de versiones para trabajar archivos en su conjunto.
  + **Git:** es una herramienta de **control de versiones** . Nos lo instalamos en nuestro ordenador, funciona por línea de comandos y nos sirve para aplicar control de versiones en cualquier carpeta albergada en nuestro dispositivo.
  + **GitHub:** es una plataforma de servicios en la nube que nos ofrece alojamiento para albergar el **repositorio central** de nuestro proyecto. Imaginemos que un equipo de cinco personas está trabajando en un mismo desarrollo. Cada uno de los ordenadores tendrá su repositorio local y, además, el equipo contará con un repositorio central alojado en GitHub, con el que irán sincronizando sus carpetas locales para que el código pueda estar distribuido entre todas las máquinas del equipo. Éste puede ser público o privado.

Para instalar Git, nos descargamos el instalador desde la página web:< [https://git-scm.com](https://git-scm.com/) > y lo ejecutamos. Debemos seguir los pasos que nos indica.

* Una vez instalado el programa, será necesario configurarlo. Las opciones de **configuración de Git** pueden existir simultáneamente en tres ámbitos distintos:
  + **Sistema:** el ámbito de una variable de sistema es todo el sistema operativo.
  + **Global:** el ámbito de una variable global es nuestra cuenta de usuario en el sistema operativo. Por tanto, una misma opción de configuración global puede tener varios valores diferentes, dependiendo de a qué cuenta de usuario esté asociada.
  + **Local:** el ámbito de una variable local es una carpeta que está siendo versionada con Git; generalmente la carpeta de un proyecto concreto.

**Ideas clave: Conociendo Git**

Git es un sistema distribuido en el que, cuando trabajamos, el código del proyecto estará repartido por varias máquinas. Es importante tener claro que cuando hacemos un *commit* lo estamos registrando en nuestro repositorio local. La mayoría de las órdenes de Git trabajan contra el repositorio local.

Cuando necesitamos **sincronizar** nuestro código con el del resto del equipo, entonces interactuaremos con el repositorio central, enviándole o pidiéndole nuevos cambios.

En Git no se guardan los cambios automáticamente. Tú decides cuándo se registra un cambio y qué contiene. Para ello, siempre debes preparar el ***commit*** utilizando el ***stage*** . El *stage* es la zona donde irás añadiendo los cambios que serán incluidos en el próximo *commit* .

El comando *git estatus* te da información valiosa sobre qué cambios están pendientes de añadirse al *stage* y cuáles están preparados para ser incluidos en el *commit* .

Cada vez que tengas que programar una nueva característica o corregir un error, abrirás una nueva rama local a partir de una de las ramas principales. Cuando hayas terminado la tarea, será el momento de verter la rama local en la rama principal. Recuerda que las ramas principales son las únicas que se comparten con las demás personas del equipo.

Una vez que hayas vertido una rama en otra, puedes borrar la primera sin miedo. No se perderán los *commits* que hayas registrado, porque ya habrán pasado a formar parte de la rama principal.

El comando *git log* te mostrará el histórico de *commits* . Aquí puedes ver los modificadores que admite, para poder personalizar su aspecto y así poder recibir la información que necesitas: < <https://git-scm.com/docs/git-log> >.

**Ideas clave: Trabajando con repositorios GitHub**

La mayoría de órdenes que lanzaremos a Git trabajarán contra nuestro repositorio local, pero en este módulo hemos visto tres órdenes clave para la **sincronización** con el **repositorio remoto** : *git fetch* , *git pull* y *git push* .

Hemos visto cómo el log nos muestra dónde están las ramas remotas (las que comienzan por origin /), pero esto siempre se basa en un caché. Para actualizar la información de este caché, utilizaremos el comando *git fetch* .

Si lo que queremos es bajarnos trabajo del repositorio que no tenemos en nuestro dispositivo, utilizaremos el comando *git pull* . Este comando hará dos cosas: actualizará la información del caché sobre ramas remotas y abocará a nuestra rama local los nuevos *commits* que haya descargado del remoto. El comando debe ir acompañado del nombre del remote y de la rama: *git pull origin master* . Si no queremos tener que escribir todo esto, podemos vincular las ramas, lanzando el comando *git branch --set-upstream-to origin/master* .

Si queremos subir al repositorio remoto los *commits* que hemos creado en local, utilizaremos el comando *git push* , acompañado del nombre del remote y de la rama ( *git push origin master* ). Si vinculamos las ramas, ya no será necesario, y podremos escribir sólo *git push* .

* Cuando creamos un repositorio, para permitir que otras personas puedan contribuir, tenemos dos opciones:
  + Las **invitamos** a través de su nombre de usuario. Cuando acepten la invitación, podrán realizar push en el repositorio.
  + Dejamos el repositorio **sin colaboradores ni colaboradoras** y quien quiera contribuir deberá realizar un *fork* del repositorio. Una vez haya subido cambios en su repositorio *fork* , podrá crear una Pull Request en nuestro repositorio, para solicitar que se incluyan sus cambios.

Las ***Pull Requests*** son un sistema muy extendido de colaboración. Mediante este sistema, alguien solicita la inclusión de sus *commits* en nuestro repositorio, pero no se incluirán hasta que no pasen un sistema de revisión. Serán las personas encargadas de la revisión las que finalmente darán el paso de incluir estos cambios en el repositorio remoto

**Otros Repositorios**

* GITLAB (Gitlab.com y Self-Managed). No limita el nº de colaboradores gratis
* Bitbucket (Cloud y Self-hosted). Es más de empresas. Limite de 5 colaboradores gratis
* Sourceforge

**Ideas clave: Otros repositorios**

En este módulo hemos visto cuáles son las ventajas e inconvenientes de utilizar GitHub como alojamiento para nuestros repositorios *online* . GitHub ha sido, durante muchos años, un referente en el mundo del *Open Source* y le han seguido de cerca plataformas como Bitbucket y GitLab. SourceForge ya no se utiliza tanto como hace años, pero sigue siendo una alternativa perfectamente válida.

Las diferencias entre GitHub y sus dos competidores próximos siempre se han basado en que GitLab y Bitbucket ofrecían repositorios privados ilimitados y un entorno de *Continuous Integration* / *Continuous Delivery* . Estas diferencias se han ido acortando desde que Microsoft adquirió GitHub y empezó a hacerle más atractivo para las empresas. Ahora las tres plataformas se asemejan bastante en cuanto a las funcionalidades que ofrecen, pero GitHub sigue siendo la que más limitaciones tiene para cuentas gratuitas.

Bitbucket es la referencia en el entorno empresarial. Su integración con herramientas como Jira y Trello y los ilimitados repositorios privados han hecho que muchos equipos de desarrollo del entorno corporativo hayan optado por esta plataforma. Ofrece un entorno de CI/CD para ayudar en los procesos de *deployment* y también tiene las típicas opciones de colaboración: *Pull Requests* y permisos de ramas.

GitLab ha sido el destino de muchas personas que salieron de GitHub después de su adquisición por parte de Microsoft. GitLab es la plataforma que ofrece menos restricciones con su cuenta gratuita y también ofrece herramientas de CI/CD y de colaboración, como *Pull Requests* (llamadas *Merge Requests* en GitLab) y permisos de ramas.

SourceForge tiene menos personas usuarias que las otras tres, pero aún sigue siendo un sitio bastante central para el mundo del *Open Source* . Si bien ofrece *Pull Requests* para entornos colaborativos, no tiene herramientas de CI/CD ni permisos de ramas.

* Pero las cuatro plataformas citadas no son las únicas. También tenemos otros servicios para alojar nuestros repositorios *online* , tales como:
  + [Microsoft Azure](https://azure.microsoft.com/)
  + [Launchpad](https://launchpad.net/)
  + [Beanstalk](https://beanstalkapp.com/) (con casi todas las opciones de pago)
  + [Cloud Source](https://cloud.google.com/source-repositories/) (sólo con repositorios privados)
  + [AWS CodeCommit](https://aws.amazon.com/codecommit/) (sólo con repositorios privados)
* Puedes ver cómo el abanico de opciones a la hora de elegir dónde residirá tu repositorio *online* es muy amplio y casi todas las plataformas te ofrecerán opciones gratuitas para que puedas probarlo y quedarte con la que más te convenga.
  + Bitbucket: < [https://bitbucket.org](https://bitbucket.org/) >
  + GitLab: < [https://gitlab.com](https://gitlab.com/) >
  + SourceForge: < [https://sourceforge.net](https://sourceforge.net/) >