

GIT

“Git (pronunciado "guit") es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente” -- Wikkipedia

Git es uno de los sistemas de control de versiones más populares de hoy en día. Gracias a su potencia y versatilidad muchos grandes proyectos de software libre están migrando sus repositorios a Git. Cada vez más es más importante saber usar Git, tanto a nivel personal como laboral. Parte del éxito de este sistema de repositorios son los repositorios online como GitHub, GitLab o Bitbucket.

Instalación de GIT en windows

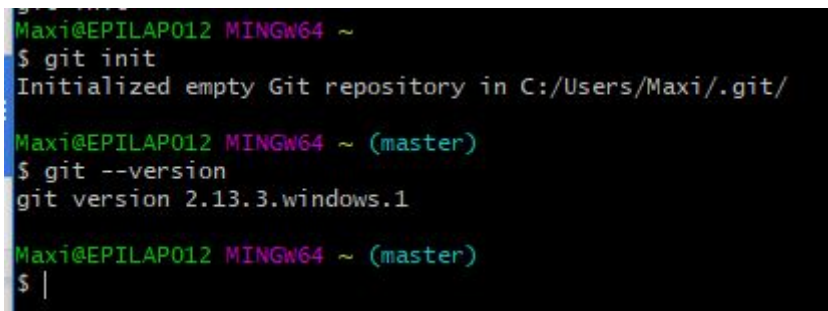
Para trabajar con GIT en windows, tendremos que bajarnos la última versión en <https://git-for-windows.github.io/>

Una vez instalado, lo primero que tenemos que hacer es configurar nuestro nombre y nuestro email para que Git pueda “firmar” nuestros commits.

```
$ git config --global user.name "Pepe Sanga"
```

```
$ git config --global user.email "pepe@educacionit.com"
```

Creando un repositorio con GIT



```
Maxi@EPILAP012 MINGW64 ~
$ git init
Initialized empty Git repository in C:/Users/Maxi/.git/

Maxi@EPILAP012 MINGW64 ~ (master)
$ git --version
git version 2.13.3.windows.1

Maxi@EPILAP012 MINGW64 ~ (master)
$ |
```

Esto quiere decir que se ha creado el directorio .git/ donde se guardará toda la información de control.

Si vamos a trabajar con Github, nos va a solicitar una clave para mantener una conexión segura con el servidor:

```

Maxi@EPILAP012 MINGW64 ~ (master)
$ git config --global user.name "Maxi Juarez"

Maxi@EPILAP012 MINGW64 ~ (master)
$ git config --global user.email "maximiliano.juarez@educacionit.com"

Maxi@EPILAP012 MINGW64 ~ (master)
$ 1
ssh-keygen -t rsa -C "your_email@yourcompany.com"

Maxi@EPILAP012 MINGW64 ~ (master)
$ 1
bash: 1: command not found

Maxi@EPILAP012 MINGW64 ~ (master)
$ ssh-keygen -t rsa -C "maximiliano.juarez@educacionit.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Maxi/.ssh/id_rsa): log
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in log.
Your public key has been saved in log.pub.
The key fingerprint is:
SHA256:yaSDNaipo4Jw7gk5KricvMLAqIcUhZd1xp++Qtqlrbo maximiliano.juarez@educacioni
t.com
The key's randomart image is:
+---[RSA 2048]-----+
| . ++                |
| . +oo               |
| o . + o             |
| . o o B .           |
|o + . + S            |
|+=. . +              |
|#+ + + .             |
|&=+o + o             |
|BO+Eo.o              |
+-----[SHA256]-----+

```

En el sitio GitHub ----> “Account Settings” > “SSH Public Keys” > “Add another public key”, denemos añadir el contenido del fichero `id_rsa.pub` generado en `/Users/your_user_directory/.ssh/`

Personal settings
Profile
Account
Emails
Notifications
Billing
SSH and GPG keys
Security
Blocked users
Repositories
Organizations
Saved replies
Applications

SSH keys

New SSH key

There are no SSH keys with access to your account.

Title

ssh educacion it

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDf72xTf1Yd3MUbrA6hVSUlpSZl851krF1H0/qh2grtgMqjKYt/4ho/1UZ
DHZCSEEEaSSnlEp35LhEi7ugQ/kKUrcRU/BkZXWgK+jFONr+gAAJOUs4H4arkckkBojy48XeoUa5j+kQDyld7blo9/
WBdAn2wzlkHmL4pDFayqhmYdjugCgx8l+RwgyzK4emVhqKg4da6lPrcN98HFUwGf8l4Kq1n/eDjmbMA/p88nnuj
UgJcqp4ldOG9PViiG5KbGZ0SbrB8tMrjVNkeNmEkMR33S3Pmpvoq0ZC0Nh5stowofBSHYO/jqcihx1+MSN6ptaR
2vQzC2uwqhWEBABWIZj5 maximiliano.juarez@educacionit.com

Add SSH key

En windows muchas veces no se crea la carpeta .ssh. Habrá que verificar si la carpeta ssh fue creada y que el archivo id_rsa.pub esta dentro.

Subiendo un proyecto a GitHub

- 1) Crear un proyecto Java.
- 2) Ejecutar git add *, para agregar todos los archivos del proyecto
- 3) Ejecutar git status para verificar lo que se enviará al repositorio
- 4) Deberíamos ver algo así:

```
Maxi@EPILAP012 MINGW64 ~/workspace2 (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   arquitectura/.classpath
        new file:   arquitectura/.project
        new file:   arquitectura/.settings/org.eclipse.jdt.core.prefs
        new file:   arquitectura/.settings/org.eclipse.m2e.core.prefs
        new file:   arquitectura/pom.xml
        new file:   arquitectura/src/main/java/arquitectura/Main.java
        new file:   arquitectura/target/classes/META-INF/MANIFEST.MF
        new file:   arquitectura/target/classes/META-INF/maven/arquitectura/arquitectura/pom.properties
        new file:   arquitectura/target/classes/META-INF/maven/arquitectura/arquitectura/pom.xml
        new file:   arquitectura/target/classes/arquitectura/Main.class
```

- 5) Ejecutar git commit -m 'Subo el proyecto a GitHub'.
- 6) git remote add origin git@github.com:maxiedit/arquitectura.git
- 7) git pull origin master
- 8) git push -u origin master

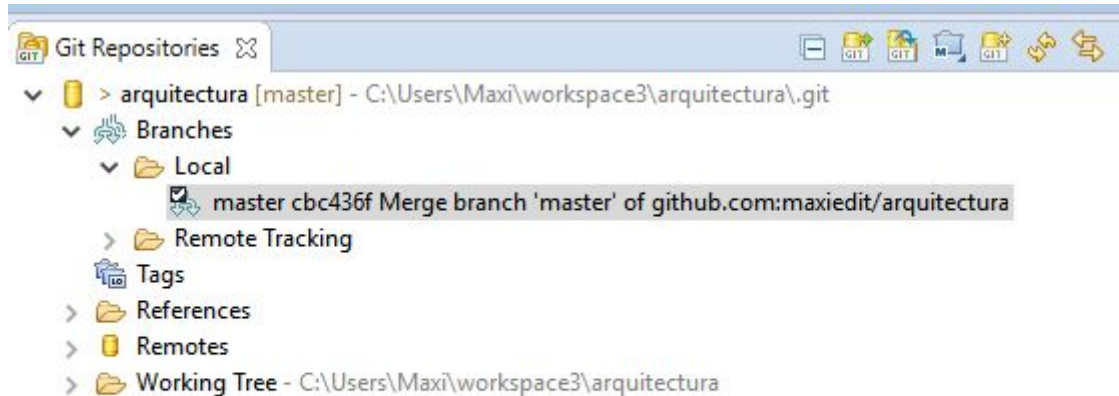
```
Maxi@EPILAP012 MINGW64 ~ (master)
$ git push origin master
Enter passphrase for key '/c/Users/Maxi/.ssh/id_rsa':
Counting objects: 27, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (17/17), done.
Writing objects: 100% (27/27), 3.16 KiB | 0 bytes/s, done.
Total 27 (delta 0), reused 0 (delta 0)
To github.com:maxiedit/arquitectura.git
   f607b86..cbc436f  master -> master
```

Clonar un proyecto de GitHub

Para clonar un repositorio online se utiliza el comando git clone:

```
Maxi@EPILAP012 MINGW64 ~/workspace3 (master)
$ git clone https://github.com/maxiedit/arquitectura.git
Cloning into 'arquitectura'...
remote: Counting objects: 30, done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 30 (delta 0), reused 27 (delta 0), pack-reused 0
Unpacking objects: 100% (30/30), done.
```

Luego se lo importa como proyecto maven.



Comandos útiles de GIT

Push a la rama actual branch: `git push origin $mi_branch`

Volver al commit anterior sin guardar los cambios: `git reset --HARD $SHA1`

Ver las ramas remotas: `git remote show origin`

Traer una rama nueva sin mergearla: `git fetch origin`

Traer una rama nueva mergeandola: `git checkout -t origin/$branch_name`

Ver todas las ramas locales: `git branch -a`

Crear una rama desde una remota: `git checkout -b $branch remotes/origin/$branch`

Crear una nueva rama desde la HEAD: `git branch $branch`

Crear una nueva rama desde la actual: `git checkout -b $new_branch $other`

Borrar la rama local: `git branch -d $branch`

Borrar la rama remota: `git push origin :$branch`

Cambiar nombre de la rama: `git branch -m $lastname $newname`

Deshacer el último commit sin pushearlo: `git reset --soft HEAD~1`

Deshacer el último commit después de haber hecho push:

`git revert HEAD`

Subir un commit parcial. Los cambios que no se han añadido permanecen estacionados y después son añadidos:

`git add $file`

`git commit -m "Comment"`

`git stash`

`git pull --rebase origin $branch`

`git push origin $branch`

`git stash pop`

Ver los commit no pusheados todavía: `git log origin/master..master`

Deshacer el último commit: `git reset --soft HEAD^`

Mezcla los últimos 10 commits en uno solo: `git rebase -i HEAD~10`

RUP

Introducción al Proceso Unificado de Desarrollo de Software

Definición

El Proceso Unificado de Desarrollo, también conocido como UP, es una metodología orientada a la construcción de software. Nace como la unión de distintas metodologías antes separadas por distintos autores, concentrando lo mejor de cada uno.

Es posible definirlo como un proceso que define quién está haciendo que, cuando lo está haciendo, y cómo alcanzar un objetivo. Es el proceso utilizado para guiar a los desarrolladores.

Se la conoce inicialmente como RUP (Rational Unified Process), que es la propuesta propietaria de la empresa Rational, y en su propuesta genérica se lo denomina UP (Unified Process).

La diferencia radical con UML es que UML es un medio, y no un fin. UML tiene como objetivo documentar, visualizar y modelar un sistema, y no define quién realiza cada actividad, tampoco define tiempos ni coordinación entre los roles de los distintos trabajadores del sistema. UML no es una metodología (como sí lo es UP) sino que es un lenguaje.

Está basado en tres aspectos claves:

- 1) está dirigido por casos de uso
- 2) está centrado en una arquitectura
- 3) es iterativo e incremental

Los tres conceptos tienen la misma importancia. La arquitectura proporciona la estructura para las iteraciones que se realizarán durante el proceso de desarrollo para evolucionar y refinar el producto, y los casos de uso definirán los objetivos y la dirección del trabajo en cada iteración.

1) Dirigido por Casos de Uso

El Proceso Unificado de Desarrollo está dirigido por el conjunto de casos de uso. Un caso de uso es una funcionalidad bien definida del sistema que proporciona al usuario un resultado. Representa a los requisitos funcionales (es decir, lo que debería hacer el sistema).

El conjunto de casos de usos constituyen el Modelo de Casos de Uso, el cual describe la funcionalidad integral del sistema. Se utilizan para guiar el diseño y la implementación, y representan el punto de partida para la construcción de los casos de prueba (test cases).

Se utiliza la palabra "dirigido" ya que los casos de uso sirven como hilo conductor del desarrollo del software.

2) Centrado en una arquitectura

El Proceso Unificado de Desarrollo está centrado en una arquitectura ya que ésta representa los cimientos del software a desarrollar, es donde el software será ejecutado. Una arquitectura comprende la definición de los siguientes conceptos:

Arquitectura de hardware

Sistema Operativo

DataBase Management System (DBMS)

Protocolos de comunicación de red

Estrategia de diseño a seguir (multicapa, orientados a servicios, etc.)

Para su definición, el arquitecto de software debe comprender los casos de uso claves del sistema. Esto se debe a que una arquitectura tiene una relación directa con los casos de uso, ya que estos deben "encajar" en la arquitectura, es decir que la arquitectura debe permitir el desarrollo normal de todos los casos de uso.

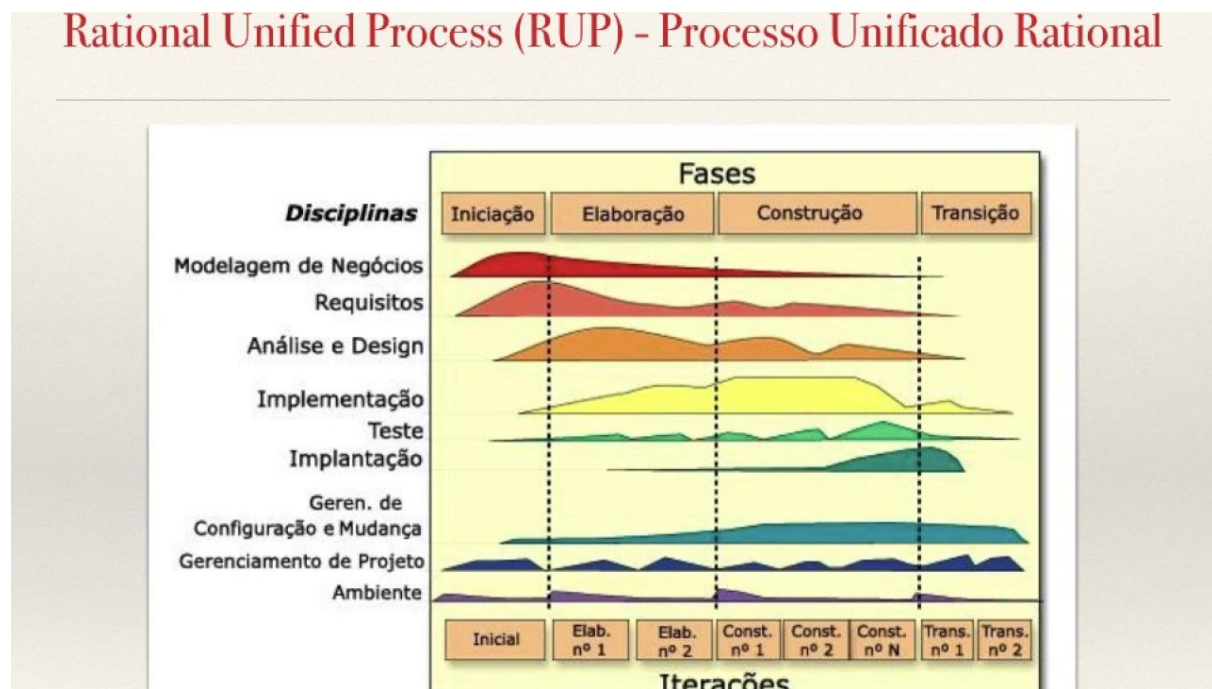
Una arquitectura debe satisfacer también a los requisitos no funcionales, como ser rendimiento, performance y velocidad de respuesta.

El objetivo es encontrar una arquitectura estable y escalable, construyéndose de tal manera que permita que el sistema de software evolucione.

3) Iterativo e incremental

El Proceso Unificado de Desarrollo es iterativo e incremental ya que divide a un proyecto en "mini-proyectos", también denominados iteraciones. El proyecto se estructura en cuatro grandes fases donde en cada fase se realizan una gran cantidad de iteraciones.

Cada iteración deberá planificarse y ejecutarse en forma controlada, basada en casos de uso bien especificados para mitigar el riesgo y logrando así en cada iteración un incremento real del sistema. Se intenta planificar todas las iteraciones de la fase (o la mayor cantidad posible) y su correspondiente orden lógico.



Cada fase termina con un hito, donde se decide si se avanza a la fase siguiente. En este punto se prevén presupuestos, planificaciones y requisitos. Se suele reunir el personal técnico con el de gestión para hacer una evaluación de la situación.

Los hitos se utilizan para estimar tiempos y recursos necesarios para la próxima fase. Sirven también para controlar el progreso con las planificaciones previamente realizadas. Si se encuentra alguna deficiencia, debe ser corregida antes de pasar a la siguiente fase.

Metodologías Ágiles

Las metodologías ágiles son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Las empresas que apuestan por esta metodología consiguen gestionar sus proyectos de forma eficaz reduciendo los costes e incrementando su productividad.

Beneficios:

Mejoran la satisfacción del cliente dado que se involucrará y comprometerá a lo largo del proyecto. En cada etapa del desarrollo se informará al cliente sobre los progresos del mismo. De ese modo, el cliente puede sumar su experiencia para optimizar las características del producto final. Se pueden evitar así numerosos malentendidos dado que el cliente poseerá en todo momento una completa visión del estado del producto.

Mejora la motivación e implicación del equipo de desarrollo. Pero esta mejora no es casual: las metodologías ágiles permiten a todos los miembros del equipo conocer el estado del proyecto en cualquier momento. Los compromisos son negociados y aceptados por todos los miembros del equipo y las ideas de cualquiera de sus integrantes son tenidas en cuenta.

Ahorran costos ya que el desarrollo ágil trabaja de un modo más eficiente y rápido que otras metodologías. Además, estos procesos ponen el foco en cumplir estrictamente el presupuesto y los plazos pactados a la hora de definir y planificar el proyecto.

Se trabaja con mayor velocidad y eficiencia. En las metodologías ágiles se trabaja realizando entregas parciales pero funcionales del producto. De ese modo, es posible entregar en el menor intervalo de tiempo posible una versión funcional del producto.

Permiten mejorar la calidad del producto. La continua interacción entre los desarrolladores y los clientes tienen como objetivo asegurar que el producto final sea exactamente lo que el cliente quiere y necesita. Además, este enfoque permite abrazar la excelencia tecnológica, lo que permite obtener un producto tecnológicamente superior.

Permiten alertar rápidamente tanto de errores como de problemas. En la etapa de planificación, el equipo ha presentado una hoja de ruta anticipando y dando respuesta a los principales problemas técnicos y a la velocidad en la que se puede trabajar. Con metodologías más tradicionales, los errores no identificados en las primeras fases del proyecto suelen acarrear costes muy altos.

Permiten rentabilizar nuestras inversiones más rápidamente. Gracias a la realización de entregas tempranas el cliente tendrá rápido acceso a aquellas funcionalidades que en verdad aportan valor acelerando el retorno de la inversión.

SCRUM

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

Promueve la innovación, motivación y compromiso del equipo que forma parte del proyecto, por lo que los profesionales encuentran un ámbito propicio para desarrollar sus capacidades.

Beneficios

Cumplimiento de expectativas: El cliente establece sus expectativas indicando el valor que le aporta cada requisito / historia del proyecto, el equipo los estima y con esta información el Product Owner establece su prioridad. De manera regular, en las demos de Sprint el Product Owner comprueba que efectivamente los requisitos se han cumplido y transmite se feedback al equipo.

Flexibilidad a cambios: Alta capacidad de reacción ante los cambios de requerimientos generados por necesidades del cliente o evoluciones del mercado. La metodología está diseñada para adaptarse a los cambios de requerimientos que conllevan los proyectos complejos.

Reducción del Time to Market: El cliente puede empezar a utilizar las funcionalidades más importantes del proyecto antes de que esté finalizado por completo.

Mayor calidad del software: La metódica de trabajo y la necesidad de obtener una versión funcional después de cada iteración, ayuda a la obtención de un software de calidad superior.

Mayor productividad: Se consigue entre otras razones, gracias a la eliminación de la burocracia y a la motivación del equipo que proporciona el hecho de que sean autónomos para organizarse.

Maximiza el retorno de la inversión (ROI): Producción de software únicamente con las prestaciones que aportan mayor valor de negocio gracias a la priorización por retorno de inversión.

Predicciones de tiempos: Mediante esta metodología se conoce la velocidad media del equipo por sprint (los llamados puntos historia), con lo que consecuentemente, es posible estimar fácilmente para cuando se dispondrá de una determinada funcionalidad que todavía está en el Backlog.

Reducción de riesgos: El hecho de llevar a cabo las funcionalidades de más valor en primer lugar y de conocer la velocidad con que el equipo avanza en el proyecto, permite despejar riesgos eficazmente de manera anticipada.

Reuniones de Scrum

a) Planificación del sprint:

El sprint es el ciclo de desarrollo del proyecto. En esta reunión, los miembros del equipo de trabajo se citan con el Scrum Master y el Product Owner y dividen el proyecto en etapas y tareas. Debe haber un responsable por cada tarea, el cual se define en función de su capacidad y una lógica estimación del esfuerzo. Esta reunión puede durar entre 4 y 8 horas.

b) Sprint diario:

Esta reunión tiene lugar cada día y no dura más de 15 minutos. En ella, cada uno de los miembros del equipo de trabajo cuenta brevemente qué hizo en la jornada pasada, lo que hará ahora y los obstáculos que ha ido descubriendo. No es una reunión descriptiva ni expositiva; se trata de ir al grano. El Scrum Master toma nota de ello, mientras el Product Owner se limita a escuchar los detalles de la evolución del proyecto (no tiene voz en esta reunión).

c) Demo del sprint:

Aunque en algunos casos se plantea como una celebración o una reunión de cierre, lo cierto es que en ella el equipo de trabajo enseña el resultado final del proyecto. En este caso, el Product Owner sí acude de forma activa: es el que se encarga de validar o no los resultados. En el caso del Scrum Master, debe tomar nota de aquellos aspectos por mejorar en caso de que los haya. A esta reunión también pueden asistir personas ajenas al proyecto en calidad de observadores.

d) Retrospectiva:

Cuando los proyectos culminan, el método Scrum plantea una última reunión, la de retrospectiva. En ella, el equipo de trabajo y el Scrum Master analizan lo que ha sido el proceso e identifican y evalúan problemas concretos. A veces se trata de problemas menores; otras, en cambio, de asuntos de fondo que tienen que ver con las dinámicas del grupo. Por ello se recomienda que a esta reunión no acudan personas ajenas al equipo de trabajo. Se estima que puede durar 2 o 4 horas y que se haga inmediatamente después del demo del sprint.