Día 1

Instalación de herrramietnas

Cmder

Git

Sourcetree

Editor de texto (Webstorm, VSCode, Sublime…)

Introducción a Git

Introducción

Qué es Git.

Puntos claves: “copia de seguridad”, prevención ante la incorporación de nuevos cambios, trabajo en equipo.

Servicios de repositorios remotos: Github, Bitbucket, …

Git por línea de commandos o con un cliente gráfico.

Instalación de herramientas/creación de cuentas

Descargar e instalar el cliente de Git.

Crear una cuenta de Atlassian.

Descargar e instalar Sourcetree (se require la cuenta de Atlassian).

Crear una cuenta de Github.

Configurar nombre e email que aparece en los commits

Esta información aparece en el archivo .gitconfig ubicado en la carpeta home del usuario del sistema. Aun así, se recomienda establecerlo mediante la línea de comandos con:

git config --global user.name “Jose Antonio Postigo”

git config --global user.email “japostigo@atsistemas.com”

Esta información es independiente de las credenciales que se usan para conectarse al repositorio.

Esta información se puede redefinir para repositorios concretos, si se ejecuta el comando desde los mismos, pero sin la opción *--global*.

Creación de un repositorio, añadir cambios al stage y commit de modificar archivo

Crear una carpeta ExplicacionRepo1 con un archivo README.md que contenga:

```
# ExplicacionRepo

Proyecto para ilustrar el uso de repositorios.
```

Hacer de ese proyecto un repositorio local con:

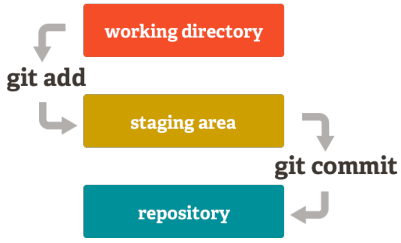
git init

Carpeta .git y estado del repositorio con:

git status

Efectuar una modificación en el archivo README.md y de nuevo consultar el estado del repositorio.

Disponemos de 3 espacios:



Añadir los cambios del archivo README.md al stage

git add README.md

Comprobar de nuevo el estado del repositorio y verificar que README.md está stageado.

Commitear los cambios

git commit

Comprobar de nuevo el estado del repositorio y verificar que no hay nada pendiente.

Abrir el proyecto con Sourcetree para ver el árbol de commits. La manera análoga para verlo en consola es:

git log

pero es poco visual y los desarrolladores, a pesar de que usen git por línea de comandas, suelen optar por cliente gráficos para visualizar el árbol de commits.

Añadir varios cambios al stage y commit de adición de archivos

Crear 3 archivos: file1.txt, file2.txt y file3.txt que contengan file1, file2 y file3 respectivamente.

Modificar el archivo README.md para describir que se han incluido los archivos.

Comprobar el estado del repositorio (hacerlo en los siguientes cambios que se efectúen):

git status

Añadir los 4 cambios al stage. Podría ejecutarse un comando *git add* por cada archivo, o en un mismo comando stagear los cambios de los 4 archivos:

git add README.md file1.txt file2.txt file3.txt

Commitear los cambios. Se puede incluir el mensaje en el mismo comando con:

git commit -m "Added file1.txt, file2.txt and file3.txt"

Abrir el proyecto con Sourcetree para ver el árbol de commits.

Sacar cambios del stage y commit de eliminación de archivos

Elimina el archivo file2.txt y comprueba el estado del repositorio con *git status*.

Como se puede observar, en el working directory queda constancia de la eliminación del archivo, sin embargo, esta eliminación no está stageada. El stageado de la eliminación de un archivo se hace igual que cuando un archivo se modifica o se añade:

git add file2.txt

Si se quiere eliminar el archivo file3.txt y además añadir ese cambio al stage, se puede usar:

git rm file3.txt

Pero hacerlo de esto forma no es lo habitual (se explica para conocer la existencia del comando).

Confirmar con *git status*.

Modificar file1.txt añadiéndole una línea y actualizar el README.md para describir el cambio sobre file1.txt y la eliminación de file2.txt y file3.txt. Luego stageamos todos esos cambios con la ejecución de un único comando:

git add .

Llegado a este punto, nos hemos dado cuenta de que en el commit que estamos preparando no queremos subir file1.txt. Habría que sacarlo del stage con:

git reset HEAD file1.txt

También tenemos que actualizar README.md para quitar la descripción del cambio sobre file1.txt. Para hacerlo basta con modificarlo y volverlo a stagear con:

git add README.md

Ya se pueden commitear los cambios:

git commit -m "Removed file2.txt and file3.txt"

Abrir el proyecto con Sourcetree para ver el árbol de commits.

Visualizar los cambios y revertir los cambios que se hicieron después del último commit

Editar README.md.

Comprobar que los cambios están pendientes de ser stageados. Podemos ver los cambios que se hicieron sobre él con:

git diff README.md

O de todos los archivos que cambiaron con:

git diff

Si hemos cometido un error en README.md que no sabemos deshacer, podemos deshacerlo según lo que viene en el último commit con:

git checkout -- README.md

También se pueden deshacer todos los cambios de todos los archivos NO stageados, hasta el último commit con:

git checkout -- .

Al comprobar el estado del repositorio se ve que no hay ningún cambio (tampoco de file1.txt, que también se cambió), y si se comprueban los archivos se observa que, efectivamente, está como estaban en el commit.

Establecer el estado del proyecto según un commit concreto

Con el working copy limpio, abrimos el SourceTree y pulsamos doble click sobre el commit al que queremos cambiar. El working copy se establecerá según dicho commit.

.gitignore y .gitkeep

Crear un archivo .gitignore para ignorar el archivo generated y el contenido de las carpetas www y dependencies.

Crear un archivo generated. Crear las carpetas www y dependencies e incluir algunos archivos en ellas.

Verificar con *git status* que estos archivos se están ignorando.

Hacer el cambio oportuno en .gitignore para que el contenido de la carpeta www sea ignorado, pero la propia carpeta forme parte del repositorio.

Stagear, mergear y pushear los cambios.

Conexión con un repositorio remoto

Crear un repositorio privado vacío en Github llamado "ExplicacionRepo-XXX", siendo XXX las iniciales del nombre del creador.

Podemos ver los repositorios remotos asociados a nuestro repositorio local con:

git remote -v

Pero en principio no habrá ninguno.

Asignamos el repositorio remoto con:

git remote add origin http://...

Normalmente se trabajará con un único repositorio remoto (origin). Pero es posible que en un proyecto se trabaje con más de un remoto. Por ejemplo: un repositorio remoto de atSistemas donde se aloje el trabajo del día a día y otro repositorio remoto del cliente donde sólo se suban las entregas.

Para subir la rama máster al repositorio remoto origin, ejecutamos el comando:

git push origin master

Como ya estamos trabajando en la rama master (porque así lo indica *git status*) y sólo hay un repositorio remoto (origin), bastaría con:

git push

Verificar en Sourcetree que ambos repositorios local y remoto están actualizados. Verificarlo también en la web de Github. Observar que el README.md se visualiza formateado desde la web.

Clonar un repositorio remoto

Para clonar un repositorio de remoto basta con ejecutar:

git clone http://... ExplicacionRepo2

Si no se indica el nombre, se tomará por defecto el que viene de remoto.

Pullear de un repositorio remoto

En el proyecto ExplicacionRepo2, editamos el archivo file1.txt. Commiteamos la modificación y la pusheamos a master.

Ahora volvemos al Sourcetree y abrimos el proyecto ExplicacionRepo1. Sólo al pulsar el botón fetch para visualizar los cambios disponibles en remoto. Comprobar la alineación de las etiquetas master y origin/master.

Actualizar el repositorio local con:

git pull origin master

Del mismo modo podría hacerse con:

git pull

De nuevo, comprobar en Sourcetree como las etiquetas master y origin/master están alineadas en el mismo commit.

Mergeo de cambios sin conflictos

En el proyecto ExplicacionRepo1, modificar el título del archivo README.md. Commitear y pushear el cambio.

En el proyecto ExplicacionRepo2, añadir a una línea de descripción al final del archivo README.md. Commitear y pushear el cambio.

Como en ExplicacionRepo2 aún no está el cambio que se subió desde ExplicacionRepo1, no nos deja subirlo hasta que no nos actualicemos.

Nos actualizamos con:

git pull

Y de nuevo lo subimos con:

git push

Sólo queda actualizar ExplicacionRepo1.

Mergeo de cambios con conflictos

Los conflictos más graves vienen de falta de sincronización en el equipo de trabajo.

En el proyecto ExplicacionRepo1, modificar una línea de la descripción. Commitear y pushear el cambio.

En el proyecto ExplicacionRepo2, modificar la misma línea de la descripción, pero con otro contenido diferente. Commitear y hacer pull de lo que hay en remoto.

Esta última operación advertiría del conflicto. En *git status* también se observa.

Cuando se abre el fichero se ve algo así:

```
<<<<<<< HEAD
(lo que tenemos en local)
=======
(lo que hay en remoto)
          >>>>>>> dddc7f0079fc82eaa2d47e5730c378f72ebbbc89
```

Hay que identificar esas líneas conflictivas, resolverlas (y por supuesto quitar los caracteres).

Una vez terminemos, se guarda y se realiza el mismo proceso que con un archivo modificado: se stagea, commitea y pushea.

Vovlemos al proyecto ExplicacionRepo1 para sincronizarnos.

Trabajando con ramas

Qué es una rama de git.

En el proyecto ExplicacionRepo1, crear una rama develop y movernos a ella ejecutando:

git checkout -b develop

Crear un archivo file2.txt que contenga file2. Stagearlo, commitearlo y pushearlo a develop.

Como la rama develop aún no está creada en el repositorio remoto, el *comando git push* a secas no funcionará. El propio git nos indica que tendríamos que usar:

git push --set-upstream origin develop

Visualizar en Sourcetree la posición de las etiquetas.

Para ver las ramas existentes en local ejecutamos el comando:

git branch

Para ver todas las ramas existentes en local y en los repositorios remotos:

git branch -a

Ahora que está el working directory limpio, podemos cambiar de rama, a master con:

git checkout master

Se puede comprobar como en esta rama aún no está el cambio de develop. Cuando se consiga una versión estable del desarrollo, se procedería a incorporar esos cambios a la rama master. Estando en ella, mergeamos lo que hay en develop ejecutando:

git merge develop

Y, por último, subir los cambios con:

git push

Stasheando cambios para cambiar de rama rápido

Cambiar a la rama develop.

Efectuamos un cambio en el archivo README.md, que stageamos, commiteamos y pusheamos.

Luego, volvemos a efectuar otro cambio distinto en el mismo archivo README.md

Resulta que por necesidades del proyecto nos piden urgentemente corregir un error de master. Nos intentamos cambiar a master.

Observamos que no se puede efectuar el cambio de rama porque tenemos trabajo en la rama develop actual. Como está a medias y no queremos commitear eso, podemos almacenar ese trabajo en una pila temporal stash (no confundir con stage).

Stageamos los cambios y los guardamos con:

git stash

En este punto, como ya está el working directory limpio, nos podemos cambiar de rama sin problema.

Cuando se hayan hecho los cambios oportunos, nos cambiamos de nuevo a develop y recuperamos el trabajo que habíamos guardado con:

git stash pop

Trabajando con la interfaz gráfica

Los siguientes pasos se harían desde la interfaz gráfica de Sourcetree:

Clonar el repositorio remoto en una nueva carpeta ExplicacionRepo3.

Cambiar a la rama a develop.

Crear un archivo file3.txt con contenido file3. Eliminar el archivo file2.txt. Modificar el contenido de file1.txt y de README.md.

Deshacer el cambio que se ha hecho sobre el README.md.

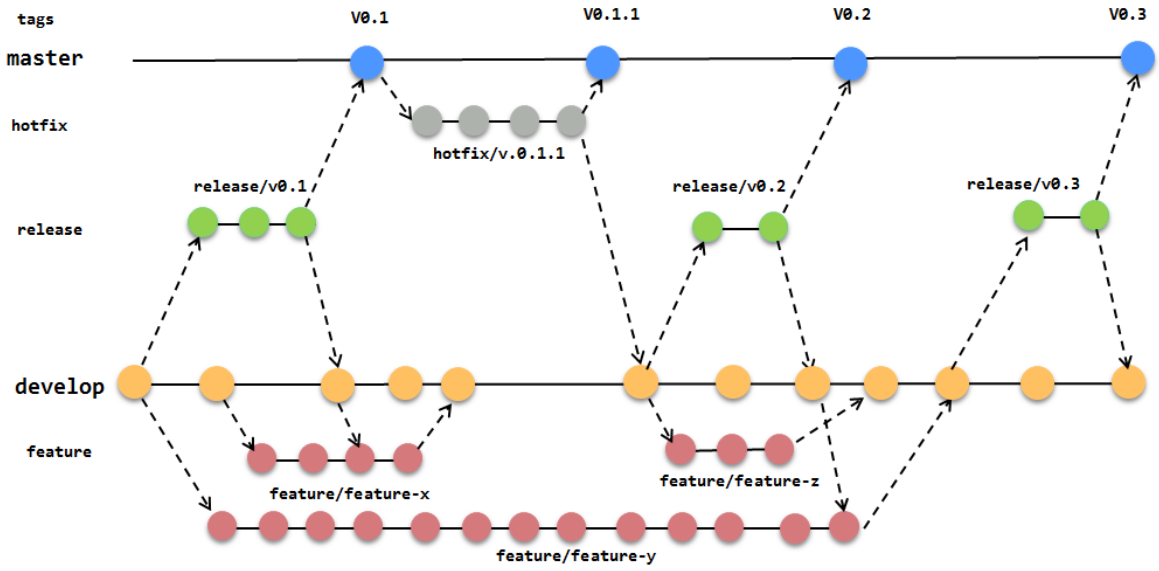
Stagear, commitear y pushear esos cambios de develop.

Cambiar la rama a master.

Mergear develop sobre master y pushear los cambios.

Abrir con Sourcetree ExplicacionRepo2 y pullear los cambios.

Git Flow



Ejercicio

Crear un fork del repositorio de la conjetura de Golbach: https://github.com/japostigo-atsistemas/GoldbachsConjecture/.

Clonar el nuevo repositorio (el fork) en local.

Cambiar a la rama develop.

Crear una rama feature/refactorization (ese es el nombre de la rama, incluyendo la barra) que parta de develop, que contendrá las modificaciones oportunas para que la variable *potentialDivisor* de la función *isPrime* pase a llamarse simplemente *i*. No subas a remoto la rama.

Cuando la rama esté lista, cambia a develop y mergea en develop el contenido de la rama feature/refactorization.

Pushea develop al repositorio remoto.

Un matemático del equipo reporta que para comprobar si un número n es primo no hace falta comprobar sus divisores hasta n – 1, si no hasta $\sqrt{n}$. Crea una rama hotfix/optimization que parta de master que incluya esta modificación. No subas a remoto la rama.

Cuando la rama esté lista, mergéala en master.

Por último, mergea en máster lo que hay en develop y pushea master al repositorio remoto.