Universidad de la República | Facultad de Ingeniería | Instituto de Computación

Taller de Programación

Consejos para el Uso de Git

1 Introducción

El objetivo de este documento es brindar algunos consejos prácticos para el uso del Software de Gestión de Cambios (SCM) de la Facultad de Ingeniería. Un SCM administra archivos y directorios y los cambios realizados en ellos a través del tiempo. Esto permite recuperar versiones antiguas de los datos o examinar el historial de cambios de estos. En particular se promueve su uso para la gestión del código fuente de las aplicaciones, aunque también puede ser utilizado para documentos generados en el proceso de desarrollo.

Git[1] es el software de gestión de cambios a utilizar. Este hace un fuerte énfasis en velocidad, soporte en la distribución, integridad de datos y el manejo de las líneas de trabajo.
Se recomienda acceder al manual de uso de Git para profundizar en varios aspectos:

http://git-scm.com/book/en/v2/Getting-Started-Git-Basics

La instalación central de Git en la Facultad de Ingeniería es GitLab (https://gitlab.fing.edu.uy/). Cada grupo contará con un proyecto de nombre `tpgrXX` (con `XX` el número de grupo) dentro del espacio de nombres de taller (`tprog`) en el cual podrá almacenar su repositorio.

Para trabajar en el entorno de desarrollo Eclipse del curso se utilizará el plug-in Egit EGit (http://www.eclipse.org/egit/) para realizar las tareas, que permitirá gestionar el versionado diréctamente sobre el IDE. En la referencia podremos ver cómo realizar tareas básicas de Git, así como hacer uso de las diferentes funcionalidades.

2 Configuración inicial

Para hacer un mejor uso de Git, es recomendable inicializar ciertos aspectos de configuración previo al proceso de trabajo.
Para esto, se recomienda tener en la raíz de nuestro proyecto dos archivos:

1. `.gitconfig`: Este archivo contiene las variables que controlan los aspectos operativos de Git. Por ejemplo, podemos incorporar herramientas para manejar las diferencias, setear colores para mejorar la visualización por línea de comandos, formato y espacios en blanco, etc.
2. `.gitignore`: Este archivo contiene línea a línea aquellos archivos que no queremos que sean considerados por Git para ser trackeados. En él se suelen incluir metadatos de los ide a utilizar, archivos de interés local, precompilados, etc.

[1] http://git-scm.com

3 Algunas definiciones previas

3.1 Repositorio

Un repositorio contiene la historia, las diferentes versiones en el tiempo y todos los distintos branch y etiquetas. En Git, cada copia del repositorio es un repositorio completo. El repositorio te permite obtener revisiones en tu copia actual.

3.2 Branch

Un branch es una línea separada de código con su propia historia (es un puntero que apunta un commit) . Es posible crear un nuevo branch a partir de uno existente y cambiar el código independientemente de otros branchs. Uno de los branchs es el original (generalmente llamado master). El usuario selecciona un branch y trabaja en este seleccionado. Este proceso de seleccionar un branch es llamado "obtener un branch" (checkout a branch). Uno de los usos más comunes es el de desarrollar las nuevas funcionalidades dentro de un branch, en lugar de hacerlo directamente en master.

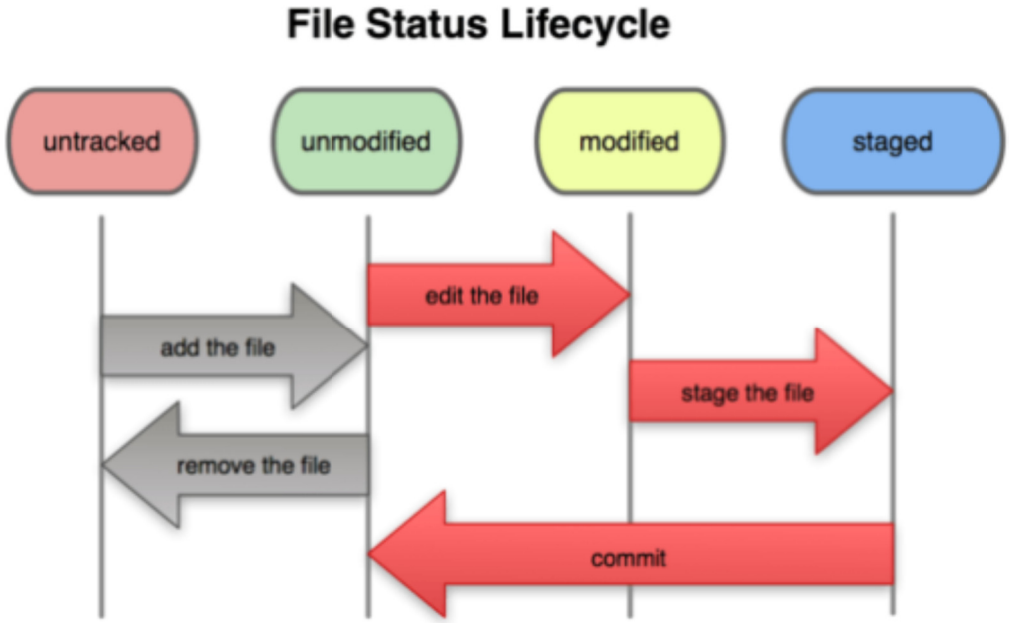
3.3 Master

Como se mencionó anteriormente, master es el nombre al branch por defecto que Git nos provee. Es recomendable no realizar el trabajo directamente sobre él sino sobre otro branch, dejándolo a éste para obtener las últimas modificaciones que se realicen en nuestro trabajo.

3.4 Commit

El comando commit de Git te servirá para confirmar algunos cambios en tu repositorio cada vez que el proyecto alcance un estado que deseas grabar.

En la siguiente imagen podemos ver cómo es el ciclo de vida de un archivo en Git.



Los archivos bajo seguimiento son aquellos que existían en la última instantánea, pueden estar sin modificaciones, modificados, o preparados. Los archivos sin seguimiento son todos los demás (cualquier archivo de tu directorio que no estuviese en tu última instantánea ni está en tu área de preparación). La primera vez que clonas un repositorio, todos tus archivos estarán bajo seguimiento y sin modificaciones, ya que los acabas de copiar y no has modificado nada.

A medida que editas archivos, Git los ve como modificados, porque los has cambiado desde tu última confirmación. Preparas estos archivos modificados y luego confirmas todos los cambios que hayas preparado, y el ciclo se repite.

3.5 Merge

Como se mencionó en el punto 3.2, el proceso de desarrollo de una nueva funcionalidad debe realizarse en otro branch, diferente a master. Luego de confirmadas todas las modificaciones, debemos incorporarlas al branch master, ya que es éste quien se encargará de consolidar todas las modificaciones de los demás integrantes de nuestro trabajo.

La forma más sencilla y básica de incorporar estos cambios a master es con el comando `git merge`. Para esto, lo que debemos hacer es movernos al branch master y ejecutar `git merge`. Este ciclo lo mostramos a continuación.

```
$ git checkout master
```

Nos movemos al branch master

```
$ git merge branch_nueva_funcionalidad
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

3.6 Tag

Al igual que otros sistemas de control de versiones, Git nos permite etiquetar momentos en la historia de nuestro trabajo que consideramos importante. Típicamente se utiliza esta funcionalidad para marcar puntos de entrega, por ejemplo, versión `v1.0`, y así.

Ejecutando el comando, `git tag`, podremos obtener un listado de todos los tags disponibles en nuestro repositorio local. Para ver cómo etiquetar, ver el punto 5.14.

4 Ciclo de trabajo

A continuación presentaremos un ciclo de trabajo típico en Git.

Lo primero que haremos será inicializar nuestros datos globales en Git, que serán el nombre y el correo electrónico. Esta información será utilizada para poder asignar el nombre de usuario y correo electrónico de cada commit y así asegurar la integridad.

```
git config --global user.name "John Connor"
git config --global user.email "jconnor@fing.edu.uy"
```

Luego,

```
$ git clone git@gitlab.fing.edu.uy:tprog/tpgrXX.git
```

Donde `tprog` es el namespace y `tpgrXX` es el nombre del proyecto creado en GitLab de Fing. Este comando nos permitirá descargar una copia del proyecto Git existente dentro de la carpeta `tpgrXX`.

Realizado el clon del repositorio, configuraremos Git según mencionamos en el punto 2. Una vez realizado lo anterior estamos en condiciones de comenzar con nuestras primeras modificaciones.

Si consideramos el escenario donde ya tenemos clonado nuestro repositorio, previo a comenzar un desarrollo, debemos realizar, considerando que estamos actualmente en el branch master:

```
$ git fetch
$ git pull origin master
```

Con este comando, obtendremos las últimas modificaciones realizadas en el branch master, así como todas las etiquetas que se encuentren en el repositorio remoto.

Ahora que tenemos todo actualizado, antes de comenzar, debemos crear un nuevo branch, de esta manera trabajaremos aislado de la rama master y de las modificaciones que en esta se introduzcan.

```
$ git checkout -b branch_name
```

Como vimos anteriormente, no solo creará el branch sino que nos movemos a él. Luego de esto, podremos realizar las modificaciones a los archivos existentes o agregar nuevos fuentes de nuestro código.

Para que estos nuevos archivos así como las modificaciones que hayamos realizado sean consideradas por Git, debemos realizar:

```
$ git add path_to_file
```

En ciertas circunstancias deseamos lo contrario, eliminar archivos trackeados por Git, para esto podemos ejecutar el siguiente comando.

```
$ git rm path_to_file
```

En cualquier momento podremos realizar un commit, ya que estos serán locales y no afectará el código en el repositorio remoto.
Los archivos nuevos o modificados considerados por el comando commit serán aquellos en los que previamente realizamos `git add`.

```
$ git commit –m "Mensaje descriptivo asociado al commit"
```

Luego, cuando hayamos terminado de desarrollar nuestra funcionalidad, debemos incorporarla a la rama master, para que el resto de los compañeros puedan obtener nuestro trabajo. Cabe mencionar que sólo debemos movernos de branch cuando tengamos todas nuestras modificaciones commiteadas.
Para esto, primero hacemos:

```
$ git checkout master
```

Nos movemos al branch master.

```
$ git pull origin master
```

Previo a incorporar las modificaciones nuestro branch a master debemos actualizarlo con las últimas modificaciones.

```
$ git merge branch_name
```

Con este último comando, incorporamos nuestras modificaciones a master.
Es importante destacar que esta operación puede traer conflictos. En caso de haber, deberemos resolverlos ejecutando el siguiente comando.

```
$ git mergetool
```

Si configuramos correctamente una herramienta para resolver conflictos, esta nos mostrará las líneas que Git no pudo resolver y debemos arreglar nosotros.

Una vez resueltos los conflictos, debemos incorporar estas modificaciones al repositorio remoto.
Esto lo realizamos con el siguiente comando:

```
$ git push origin master
```

Con este último comando, el resto del equipo podrá descargar nuestras modificaciones como lo mencionamos en los primeros pasos.

Finalmente, podremos etiquetar nuestro último commit, por ejemplo, para marcar un punto en la historia donde queramos tener un entregable. Estás etiqueta siempre deberán ser sobre un commit de master.

Por lo tanto, parados en el branch master realizamos:

```
$ git tag –a entrega1 –m "Entrega final de la tarea 1"
```

Luego, debemos subir esta nueva etiqueta al repositorio remoto, siendo este último punto importante ya que sino esta quedará visible localmente. Para ello realizamos:

```
$ git push origin entrega1_tpgr00
```

5 Usos en Git

5.1 Cómo clonar un repositorio

Si deseamos tener una copia de un repositorio Git existente, precisamos realizar un `git clone`. Con este comando recibiremos una copia completa de lo que se encuentra en el repositorio. Ejemplo de uso:

```
$ git clone https://gitlab.fing.edu.uy/tprog/tpgrXX.git
```

Donde `tprog` es el namespace y `tpgrXX` es el nombre del proyecto creado en gitlab de fing.

5.2 Cómo agregar nuevos archivos para que sean trackeados por Git

Para incorporar nuevos archivos a Git y de esta manera poder incluirlos a nuestro árbol de trabajo o para preparar el contenido a un nuevo commit, utilizamos el comando `git add`.

```
$ git add path_to_file
```

5.3 Uso de git status

La ejecución del comando `git status` nos permitirá obtener un resumen de qué archivos tienen cambios, en cuál branch me encuentro y cuáles son los archivos prontos para ser commiteados.

```
$ git status
```

5.4 Cómo ignorar ciertos archivos que no quiero que sean trackeados

Muchas veces tenemos ciertos archivos que no deseamos sean incluidos en nuestro árbol de trabajo, como por ejemplo metadatos de los IDE de desarrollo, archivos del sistema o log de información. Para estos casos podremos crear un archivo de nombre `.gitignore`.
En el podremos poner aquellos archivos, línea por línea, que no deseamos sean incluidos en nuestro árbol de trabajo y por lo tanto descartados para el control de cambio.
Ejemplo:

```
$ cat .gitignore

*.ide
*~
```

5.5 Cómo confirmar cambios ("commitear")

Una vez definidos los cambios a realizar y que nuestros archivos se encuentran en el área `staged`, podemos confirmarlos. Tener en cuenta que aquellos archivos modificados o creados recientemente y que no se les haya ejecutado `git add` no se considerarán para el commit. Por lo tanto, el comando Git para realizar este commit es el siguiente:

```
$ git commit –m "Mensaje descriptivo asociado al commit"
```

5.6 Cómo quitar y mover archivos

Para eliminar un archivo de nuestro árbol de trabajo y que este no sea considerado en un futuro debemos ejecutar el comando:

```
$ git rm path_to_file
```

A diferencia de otros sistemas de gestión de cambio, Git no lleva un track de las modificaciones de un archivo. Esto es, si renombramos un archivo, Git no guarda ningún metadato que indique esta modificación. Sin embargo, Git provee un mecanismo para hacer esto viable.

```
$ git mv old_file_name new_file_name
```

5.7 Cómo visualizar el histórico de cambios

Luego de crear varios commit, o si nos clonamos un repositorio con varios commit, podremos ver el histórico de modificaciones (commits con su descripción), ejecutando el siguiente comando:

```
$ git log
```

5.8 Cómo traer archivos remotos

Si ejecutamos:

```
git fetch
```

Este comando va hacia el repositorio remoto y descarga toda la información que no poseemos. Luego de hacer esto se puede tener referencia a todos los branchs y etiquetas.
Es importante notar que `git fetch` trae la información desde el repositorio remoto, pero no incorpora estos cambios a mi repositorio local, sino que debemos realizarlo nosotros manualmente cuando estemos preparados.

En caso de tener localmente un branch que sigue un branch remoto, podremos hacer uso de `git pull` que automáticamente nos realizará el `fetch` y luego un `merge` de los datos remotos con los nuestros locales.

5.9 Cómo subir cambios al repositorio remoto

Cuando tenemos modificaciones ya commiteadas de nuestro proyecto y deseamos compartirlas, debemos subirlas a nuestro repositorio remoto.
El comando para esto es:

```
$ git push origin branch_name
```

Siendo `branch_name` el nombre del branch donde estamos trabajando y realizando las modificaciones

5.10 Cómo listar branchs

Para conocer y listar los branchs locales debemos ejecutar:

```
$ git branch
```

y en caso de querer listar los locales así como los branchs remotos, debemos ejecutar:

```
$ git branch –a
```

5.11 Cómo crear un branch nuevo

Git posee un branch por defecto denominado master. Cuando comenzamos a realizar commits sobre este branch, Git automáticamente nos moverá hasta este último punto. De esta manera, un branch lo podemos ver como una secuencia en el tiempo de commits sobre nuestros archivos.

Un ciclo de trabajo recomendado es realizar las modificaciones de nuestro trabajo en un nuevo branch, diferente a master. Para hacer esto realizamos:

```
$ git checkout -b branch_name
```

Con este comando hemos realizado un nuevo branch y nos hemos movido a él, es decir, de ahora en más, todas las modificaciones y commits que realizemos se harán sobre este nuevo branch.

5.12 Cómo incorporar cambios a otro branch

Supongamos el escenario del punto anterior, donde nos hemos movido a un nuevo branch `branch_name`. A su vez, consideremos que tenemos cambios ya commiteados en este nuevo branch y que deseamos incorporarlos a la rama master. Para esto, el ciclo es el siguiente:

`$ git checkout master` : Nos movemos a la rama master y realizamos

```
$ git merge branch_name
```

Este último paso podrá resultar en conflicto o no dependiendo de los archivos modificados. En caso de encontrar conflictos Git lo mencionara indicando qué archivos se deben modificar para culminar con éxito la operación.

Una alternativa para resolver estos conflictos es con el uso de herramientas gráficas. Para esto ejecutar:

```
$ git mergetool [2]
```

5.13 Cómo subir cambios al repositorio remoto

Una vez incorporado los cambios a la rama master estamos en condiciones de poder compartirlo con el resto de los integrantes de mi grupo de trabajo. Para esto, debemos ejecutar:

```
$ git push origin master
```

Siendo `master` el branch remoto donde incorporaremos nuestros cambios.

[2] Para poder ejecutar este comando debemos tener configurada una herramienta para la resolución de conflictos. Ver http://meldmerge.org/help/resolving-conflicts.html en caso de usar meld.

5.14 Cómo etiquetar

Una vez culminado nuestro trabajo podemos etiquetarlo. Una etiqueta es un punto específico que consideramos importante, por ejemplo, cuando queremos marcar una versión release.

Para etiquetar nuestro último commit realizamos:

```
$ git tag –a entrega1 –m "Entrega final de la tarea 1"
```