Git y Github

Tutorial sobre control de versiones

Contenido

1 Control de versiones....2
2 Git y Github....2
3 Conceptos y operaciones....2
4 Trabajo diario....3
5 Git, secuencia de trabajo....4
Instalar Git....4
Git, obtener ayuda....4
Configuración....4
Crear repositorio....4
Crear revisiones....4
Desarrollo en ramas....4
6 Github, secuencia de trabajo....5
Cuenta en Github....5
Crear un repositorio en Github....5
Autenticación en Github....5
Asignar un repositorio remoto....6
Actualizar repositorio remoto (push)....6
Editar en Github....6
Actualizar repositorio local (pull)....6
Actualizar cambios locales en el repositorio remoto....7
Ramas en Github....7
7 Replicar un repositorio remoto: git clone....7
8 Ignorar archivos: .gitignore....8
9 Bibliografía....8

Resumen. *En proyectos de desarrollo incremental, ya sean de autoría colectiva o individual, es conveniente retener el estado del material en un momento específico. Este estado se denomina versión o revisión, y se identifica con un nombre o un número. Es posible así experimentar libremente opciones potencialmente disruptivas, con la certeza de poder retornar a la versión guardada. Git es un software de control de versiones instalable en una máquina local apto para el desarrollo colaborativo sincronizado a través de un servidor remoto en Internet o en una red local, o aún en la propia máquina. Github es una plataforma de Internet basada en Git que actúa como servidor remoto en Internet. Este instructivo explica el manejo de Git y Github para el desarrollo de software, documentación, u otros tipos de proyecto.*

1 Control de versiones

En Ingeniería de Software, *control de versiones* designa los sistemas encargados de manejar los cambios realizados en programas, documentos, sitios web u otras colecciones de información. Los cambios pueden realizarse en distintos archivos, en distintos momentos y por diferentes personas.

Para rastrear estos cambios, se definen periódicamente estados o versiones, identificados por un nombre, e.g. "Rev1.1", "GUI-v2". Una *versión*, *revisión* o *edición* retiene el estado de todos los documentos en un determinado momento, con el registro de todos los cambios ocurridos desde la versión anterior, así como su autoría (quien de los usuarios autorizados lo realizó). El sistema de control de versiones permite comparar distintas versiones, volver a una versión determinada, o fusionar dos versiones distintas en una sola incorporando todos los cambios y asegurando su coherencia. Esto permite realizar cambios o generar extensiones sin riesgos, ya que es posible volver a una versión anterior si en la nueva aparecen errores [1].

Un *sistema de control de versiones distribuído* es un control de versiones en el cual el contenido completo del código o los documentos se replica en el computador de cada colaborador [2]. Esto permite la creación de una o más versiones en paralelo y su combinación incorporanto todos los cambios. Una versión paralela a la principal se denomina *rama* (*branch*) , y su integración con la rama principal se denomina *fusión* o *combinación* (*merge*) [3].

2 Git y Github

Git es un sistema de control de versiones distribuído: un directorio Git en un computador replica todo el contenido, historia y versiones del conjunto de archivos de un proyecto, independientemente del acceso a una red o un servidor central. Originalmente creado por Linus Torvalds en 2005, es software libre de código abierto licenciado bajo GPL-2.0-only [4].

Github es un servicio de alojamiento en Internet para el desarrollo de software y control de versiones basado en Git. Además del control de versiones distribuído provisto por Git, Github provee control de acceso, rastreo de errores (*bug tracking*), solicitudes de ampliaciones del software, manejo de tareas, integración continua y wikis para cada proyecto. Es muy usado para proyectos de desarrollo de software de código abierto [5].

Otros servicios de alojamiento para control de versiones son Bitbucket y Gitlab.

3 Conceptos y operaciones

En los sistemas de control de versiones, un *repositorio* es una estructura de datos con los metadados necesarios para retener la historia de los cambios realizados sobre un conjunto de archivos o una estructura de directorios. En los sistermas distribuidos como Git, los metadatos se replican en el sistema de cada usuario; en los sistemas centralizados como Subversion se guardan en un servidor central [6].

Un repositorio existente se puede *clonar* hacia la máquina local desde un servidor remoto, creando un repositorio local idéntico al repositorio remoto.

Un *área de ensayo* (*staging area*) es un almacenamiento intermedio entre un conjunto de datos fuente y un conjunto destino [7]. En control de versiones, en esta área se retienen todos los archivos de los cuales se quiere realizar el seguimiento; el conjunto origen son los datos en su estado actual, el conjunto destino una *versión* o *revisión*, el estado de los datos en cierto momento. La operación de consolidar los datos del área de ensayo en una nueva revisión se denomina *commit* (comprometer, remitir) [8]. Al realizar el *commit* se debe adjuntar un mensaje descriptivo de esa versión. Los datos grabados en una versión tienen caracter permanente; se puede retornar a ellos si se lo desea.

Para continuar el desarrollo sin comprometer el estado de revisión actual, es posible crear una *rama* o *branch*, una duplicación de los archivos en su estado actual que puede ser modificada sin comprometer los archivos de la rama principal (*trunk*, tronco). Una vez asegurada la calidad del trabajo en la rama secundaria, se la puede combinar con la rama principal mediante la operación

denominada *merge* o *fusión*. La selección de la rama donde se desea trabajar se realiza mediante la operación denominada *checkout*, que equivale a activar una rama determinada.

La figura muestra un esquema del desarrollo usando ramas. Cuando una versión se considera estable, apta para su uso y publicación, es posible asignarle una etiqueta o *tag*; esta etiqueta permite a los usuarios identificar las versiones recomendadas para usar en producción.

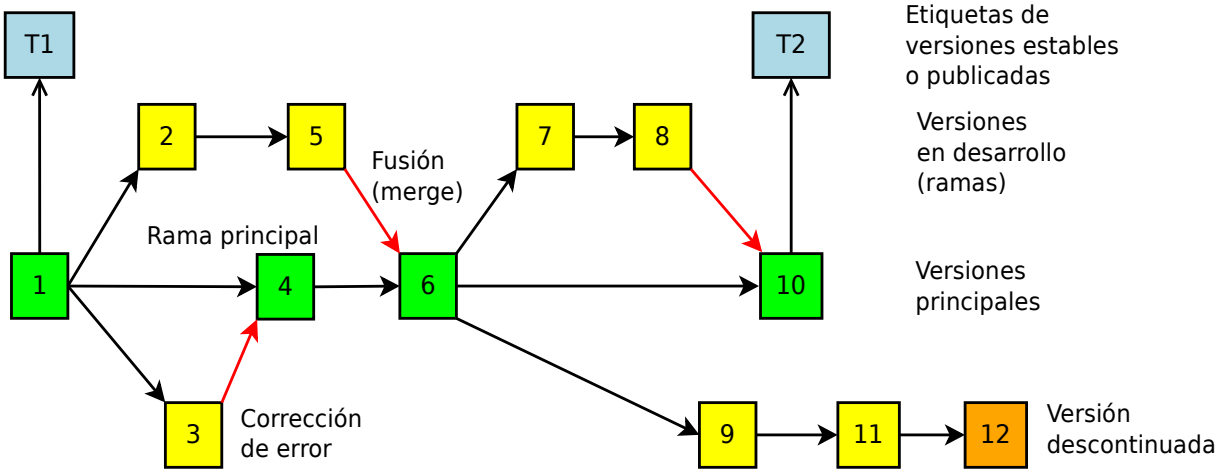

Figura 1: Grafo de control de versiones con rama principal, ramas secundarias en desarrollo o de correcciones, fusiones periódicas, y asignación de etiquetas a las versiones estables.

4 Trabajo diario

Una vez creados el repositorio Git local y el repositorio remoto en Github, la secuencia de operaciones habituales en una sesión de trabajo es la siguiente:

1. Actualizar el repositorio local desde el remoto (`pull`):
   ```
   $ cd <directorio local>
   $ git remote show main  # muestra estado de sincronización
   $ git pull origin main  # actualiza repositorio local desde el remoto
   ```
2. Trabajo en el directorio local: modificar, agregar o eliminar archivos.
3. Preparar el envío de modificaciones locales (`commit`):
   ```
   $ git add <nvo_arch_1> <nvo_arch_2> ....     # archivos nuevos a rastrear
   $ git commit -a -m "descripción del commit"  # incluye todo lo modificado
   ```
   Los archivos nuevos requieren `git add`; la opción `-a` del comando `git commit` incluye todos los archivos existentes modificados.
4. Actualizar repositorio remoto (`push`) desde el repositorio local:
   ```
   $ git push origin main
   ```
   Pide un código de autenticación; en la sección Autenticación en Github se explica cómo crearlo.

5 Git, secuencia de trabajo

Instalar Git

En Linux, Git suele estar disponible como paquete estándar en la distribución. En la página de descargas de Git se ofrecen paquetes de instalación para Linux, MS Windows y macOS.
Para usar Git necesitaremos un intérprete de comandos. En MS Windows, se puede usar Git bash, que viene incluido en Git para Windows. Para Mac y Linux se puede usar un terminal. El siguiente comando verifica si Git está instalado; obtendremos una salida similar a la mostrada:

```
$ git --version        # para verificar si Git está instalado
git version 2.25.1
```

El texto después de `#` es un comentario, no necesita escribirse. Los textos entre signos `<...>` deben sustituírse por los valores correspondientes según el contexto; por ejemplo, en `git help <comando>`, escribir `git help clone` para obtener ayuda sobre el comando `clone` de Git.

Git, obtener ayuda

Para obtener información sobre los comandos de Git:

```
$ git help --all        # muestra todos los comandos de git y su descripción
$ git help <comando>    # muestra ayuda del comando indicado
$ git <comando> --help  # hace lo mismo
```

Configuración

Establece el nombre de usuario y su dirección e-mail, datos necesarios para el rastreo de versiones. En el siguiente ejemplo, cambiar por nombre de usuario y e-mail, pero conservar las comillas:

```
$ git config --global user.name "w3schools-test"
$ git config --global user.email "test@w3schools.com"
```

La opción `--global` configura Git para todos los repositorios en una misma máquina; si se omite, asigna estos datos solo al repositorio actual.

Crear repositorio

Para crear un repositorio local:

```
$ mkdir <nombre_proyecto>
$ cd <nombre_proyecto>
$ git init       # inicializa el directorio local como repositorio Git
```

Ahora es posible crear archivos y directorios en el repositorio, pero estos aún no han sido registrados para mantener su rastreo. Para ver el estado del repositorio:

```
$ git status     # los archivos aún no tienen seguimiento, están "untracked"
```

Crear revisiones

Para registrar los archivos a realizar seguimiento (*staging*):

```
$ git add <nombre_archivo> ...   # ahora estos archivos tendrán seguimiento
$ git status     # los muestra como nuevos archivos
$ git add --all  # agrega todos los archivos en el directorio
```

Para preparar una nueva revisión guardando todos los archivos en su estado actual (*commit*):

```
$ git commit -m "mensaje indicando estado, modificaciones, etc"
$ git commit -a -m "mensaje..."   # -a o --all incluye todo lo modificado
$ git log   # muestra la historia de "commits", las versiones guardadas
```

La opción `--all` o `-a` realiza simultáneamente las operaciones de *staging* y *commit* sobre todos los archivos modificados recientemente; no incluye los archivos nuevos, que precisarán un `git add`.

Desarrollo en ramas

Crear una rama, mostrar las ramas existentes, activar una rama para trabajar en ella:

```
$ git branch <nombre_nueva_rama>  # crea una nueva rama
$ git branch   # muestra las ramas existentes, con asterisco la activa
* main
  <nombre_nueva_rama>
$ git checkout <nombre_nueva_rama>      # activa la nueva rama
Switched to branch '<nombre_nueva_rama>'
$ git branch                      # muestra asterisco en la rama activa
  main
* <nombre_nueva_rama>
```

Ahora todas las modificaciones y nuevos archivos estarán solo en la nueva rama, sin afectar la principal (usualmente llamada `main`). Es posible crear una versión en la nueva rama con los comandos `git add` y `git commit` como ya se indicó.
Una vez aceptados los cambios en la nueva rama, es posible realizar la combinación o fusión (*merge*) con la rama principal:

```
$ git checkout main             # activa rama principal
Switched to branch 'main'
$ git merge <nombre rama>       # combina esta rama con la principal
Updating ...
$ git branch -d <nombre rama>  # borra la rama, ya integrada a la principal
Deleted branch '<nombre_rama>' ...
```

Si un mismo archivo ha sido modificado en dos ramas distintas, el comando `git merge` indicará un conflicto; deberá modificarse el archivo en una u otra rama para hacerlos coincidir en contenido antes de poder concretar la fusión.

6 Github, secuencia de trabajo

Cuenta en Github

Para poder usar una plataforma remota de control de versiones es preciso crear una cuenta en esa plataforma. Para Github, en https://github.com/ botón *Sign up*; usar la misma dirección de correo electrónico usada en `git config`.

Crear un repositorio en Github

Ingresar al sitio de Github con la cuenta recién creada. En la barra de menú superior, a la derecha, con el signo de + elegir *Repositories*, *New*; indicar el nombre del repositorio local, llenar los otros campos del formulario:

- *Description*: una descripción breve del repositorio.
- *Public* muestra los archivos a quienquiera; *Private* solo a quienes se autorice. En ambos casos, los colaboradores deben ser autorizados.
- *Add a README file*: agrega un archivo de nombre `README.md` para escribir una descripción detallada del proyecto. Recomendado.
- *Add .gitignore*: para registrar los tipos de archivo que no tendrán seguimiento en el archivo `.gitignore`.
- *Choose a license*: permite elegir entre varias licencias para proteger el material. Recomendadas: para software, elegir *GNU General Public License v3.0*; para documentos, cursos y tutoriales, *Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional*.
- *Create repository* para crear el nuevo repositorio. Quedará accesible en `https://github.com/<usuario>/<repositorio>`. Este será el URL del repositorio remoto en Github.

Autenticación en Github

En el primer intento de actualización desde el repositorio local, Github pedirá alguna forma de autenticación. Una de ellas es el *Private Access Token (PAS)*, un código que será solicitado al realizar una operación `push`. Para crear un nuevo *token*:

1. En la esquina superior derecha de cualquier página, clic sobre la foto de perfil, luego *Configuración*, *Settings*.
2. En la barra lateral izquierda, *Ajustes de desarrollador*.
3. En la barra lateral izquierda, *Tokens de acceso personal* (está bien al final).
4. Clic en *Generar un nuevo token*. Asígnar al token un nombre descriptivo.
5. Elegir "repo" para obtener permisos de manejo del repositorio.

Para ver los *tokens* propios: https://github.com/settings/
Información sobre *tokens*: https://docs.github.com/es/

Asignar un repositorio remoto

Para asignar un repositorio remoto contra el cual se sincronizará el repositorio local:

```
$ cd <directorio_local>    # ir al directorio del repositorio local
$ git remote add origin https://github.com/<usuario>/<repositorio>
$ git remote -v  # muestra el respositorio remoto agregado como origen
origin  https://github.com/<usuario>/<repositorio>  (fetch)
origin  https://github.com/<usuario>/<repositorio>  (push)
```

Actualizar repositorio remoto (`push`)

La réplica de en el servidor remoto se realiza con la operación denominada *push* (empujar). Para replicar en el repositorio remoto el contenido del repositorio local,

```
$ git push --set-upstream origin main
Enumerating objects: ...
```

La operación `push` replica la rama principal local en el URL establecido como origen, y lo establece como la rama remota por defecto. Para ver el estado final:

```
$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.
```

Editar en Github

Es posible editar y crear nuevos archivos en línea accediendo con la cuenta propia al sitio de Github. Luego de elegir un archivo, el botón del lápiz permite editar; al finalizar, guardar los cambios con *Commit changes*, agregando una descripción de lo hecho en esta edición. El botón *Add file* permite agregar un nuevo archivo en el repositorio remoto.

Actualizar repositorio local (`pull`)

Para actualizar el repositorio local a una revisión posterior del servidor remoto, se usa la operación *pull* (tirar).

```
$ git status      # indica si el local está retrasado y en cuantos commits
En la rama main
...
$ git pull origin  # actualiza el repositorio local desde el remoto
...
```

El comando `pull` combina los comandos `fetch` y `merge`. La misma operación anterior en forma más detallada sería:

```
$ git fetch origin  # muestra lo que ha cambiado en Github
Desde https://github.com/<usuario>/<repositorio>
...
$ git status  # indica si el local está retrasado y en cuantos commits
En la rama main
...
$ git log origin/main    # muestra commits anteriores
$ git diff origin/main   # diferencias entre main local y remoto
$ git merge origin/main  # combina repositorio main local con remoto
```

Actualizar cambios locales en el repositorio remoto

Luego de realizar cambios en el local, para actualizar el repositorio remoto:

```
$ git commit -a -m "mensaje de actualización"  # prepara la versión nueva
$ git status  # muestra el retraso de remoto en 1 commit
$ git push origin  # envía los cambios al origin remoto
```

Ramas en Github

Crear una nueva rama en Github. En el sitio de Github, el botón de ramas, donde indica *main*, permite crear una nueva rama escribiendo un nombre en la caja de texto *Find or create a branch*. La rama queda activa, con su nombre visible en el botón de ramas. De ahora en adelante, los cambios quedarán en la nueva rama, sin afectar la rama principal *main*.

Actualizar Git local con nueva rama. Para actualizar el repositorio Git local incorporando la nueva rama del repositorio Github remoto:

```
$ git status         # todo actualizado, en rama main
$ git branch         # muestra ramas en local, solo rama main, activa
$ git branch -a      # muestra ramas locales y remotas, asterisco en activa
$ git branch -r      # muestra solo ramas remotas
$ git checkout <nueva_rama>  # activa rama remota
$ git pull           # actualiza rama local desde remota
$ git branch         # muestra ramas locales, con <nueva_rama> activa
```

Crear rama local y replicar en repositorio remoto. Para crear una nueva rama en el repositorio Git local, y luego actualizar el repositorio remoto en Github con la nueva rama local:

```
$ git branch <nueva_rama_local>      # crea nueva rama local
$ git checkout <nueva_rama_local>    # activa nueva rama local
```

Luego de realizar cambios en archivos locales y/o agregar archivos:

```
$ git status        # muestra archivos no agregados, "not staged for commit"
$ git add <nombre archivo>  # agrega para commit, pueden ser varios
$ git status        # muestra los cambios para agregar al commit
$ git commit -m "archivos modificados en rama local"
$ git push origin <nueva_rama_local>
```

Incorporar cambios en Github. En el repositorio remoto de Github se puede ver que el repositorio tiene una nueva rama.

- El botón *Compare & pull request* permite ver los cambios realizados en los archivos existentes y los archivos nuevos agregados.
- Si los cambios realizados se consideran satisfactorios, el botón *Create pull request* permite crear una solicitud de actualización y ver los cambios propuestos.
- Los cambios se pueden incorporar con el botón *Merge pull request*.
- Una vez realizada fusión (*merge*), se puede borrar la rama ya fusionada con el botón *Delete branch*.

7 Replicar un repositorio remoto: `git clone`

Para obtener una copia completa de un repositorio remoto de Github en la máquina local:

1. En el sitio del repositorio en Github, obtener el URL, e.g.
   `https://github.com/vagonbar/pythonbas`
2. En la máquina local, trasladarse al directorio donde se quiere replicar el repositorio y emitir el comando `git clone` con el URL correspondiente:
   `$ git clone https://github.com/vagonbar/pythonbas`
3. Se crea un subdirectorio con el nombre del proyecto, en este ejemplo `pythonbas`:
   `$ cd pythonbas   # ubica en el nuevo repositorio Git local`

8 Ignorar archivos: `.gitignore`

El archivo `.gitignore` permite registrar patrones de nombres de archivos y directorios que no se quieren compartir, como archivos de log, archivos temporales, notas, borradores, u otros. Un ejemplo del contenido de un archivo `.gitignore` puede ser:

```
.gitignore
.git
```

```
notas
__pycache__
*.bak
*.log
*.pyc
*.pyo
*~
```

9 Bibliografía

Tutoriales y material complementario.

- W3 Schools. Git Tutorial, https://www.w3schools.com/git/. Un tutorial simple para aprender mediante ejemplos [9].
- Open Source Guides. *Starting an Open Source Proyect*. https://opensource.guide/starting-a-project/. Una guía para iniciar proyectos de código abierto con Git y Github [10].
- Scott Chacon, Ben Straub. *Pro Git*. 2nd ed (2014). En inglés, disponible en línea: https://git-scm.com/book/en/v2 [11]. Un manual exhaustivo sobre el uso de Git.

Referencias

[1] Wikipedia. *Version control*. 2022. https://en.wikipedia.org/wiki/Version_control.

[2] Wikipedia. *Distributed version control*. 2022. https://en.wikipedia.org/wiki/Distributed_version_control.

[3] Wikipedia. . 2022. https://en.wikipedia.org/wiki/Merge_(version_control).

[4] Linus Torvalds, Junio Hamano, y otros. . 2022. https://git-scm.com/.

[5] GitHub, Inc. *GitHub*. 2022. https://github.com.

[6] Wikipedia. *Repository (version control)*. 2022. https://en.wikipedia.org/wiki/Repository_(version_control).

[7] Wikipedia. *Staging (data)*. 2022. https://en.wikipedia.org/wiki/Staging_(data).

[8] Wikipedia. *Commit (version control)*. 2022. https://en.wikipedia.org/wiki/Commit_(version_control).

[9] W3 Schools. *Git Tutorial*. 2022. https://www.w3schools.com/git/default.asp.

[10] Open Source Guides. *Open Source Guides*. 2022. https://opensource.guide/.

[11] Scott Chacon, Ben Straub. *Pro Git*. 2nd Ed, 2022. https://git-scm.com/book/en/v2.
