 

Mochila del viajero

Práctica individual obligatoria I, consigna 2

Curso Introducción a la informática

Docente Jorge Agustín Pereyra

Estudian Alan Muñoz Badillo

**Cheatsheet Git**

|  |  |
| --- | --- |
| **Creación del repositorio** | |
| git init | Inicia un repositorio vacío en la carpeta local en que se ejecuta. |
| git clone “url del repositorio remoto a clonar” | Clona, en local, un repositorio almacenado en algún sistema de control de versiones dispuesto en la nube. |
| **Configuraciones** | |
| git config user.name “nombre de usuario” | Configura el nombre de usuario para el desarrollo del proyecto en particular vinculado a git. Se utiliza para identificar quién crea, modifica, elimina elementos del repositorio, así como la vinculación a la nube de estos. Para trabajar en la nube procure coincidir con los datos de la cuenta utilizada en dichos servicios. |
| git config user.mail “email del usuario” | Configura el email de usuario para el desarrollo del proyecto en particular y vinculado a git, se utiliza para identificar quién crea, modifica, elimina elementos del repositorio, así como la vinculación a la nube de estos. Para trabajar en la nube procure coincidir con los datos de la cuenta utilizada en dichos servicios. |
| git config --global user.name “nombre de usuario” | Configura el nombre de usuario para el desarrollo de todos los proyectos creados en la computadoras y vinculados Git, se utiliza para identificar quién crea, modifica, elimina elementos del repositorio, así como la vinculación a la nube de estos. Para trabajar en la nube procure coincidir con los datos de la cuenta utilizada en dichos servicios. |
| git config –global user.mail “email del usuario” | Configura el email de usuario para el desarrollo de todos los proyectos creados en la computadoras y vinculados Git, se utiliza para identificar quién crea, modifica, elimina elementos del repositorio, así como la vinculación a la nube de estos. Para trabajar en la nube procure coincidir con los datos de la cuenta utilizada en dichos servicios. |
| **Gestión de los archivos** | |
| git add “Nombre del archivo” | Añade el archivo correspondiente a la “Staging Area” (memoria ram) y los deja a la espera para ser subidos al repositorio local. Siempre y cuando no se encuentre trakeado. |
| git add . | Añade todos los archivos que no se encuentren trakeados a la “Staging Area” (memoria ram) y los deja a la espera para ser subidos al repositorio local. |
| git rm –cached “Nombre del archivo” | Elimina un archivo tanto del repositorio local (realizado ya el “commit”) como de la “Staging Area”, pero lo mantiene en el directorio de trabajo. Si se elimina del repositorio local, queda sin seguimiento, y si se realiza un “add nombre del archivo” pasa directamente al repositorio local, saltándose la “Staging Area”. |
| git restore –staged “Nombre del archivo” | Elimina un elemento presente en la “Staging Area” dejando de hacerle seguimiento. |
| git restore “Nombre del archivo” | Afecta a archivos modificados en el directorio de trabajo, si hay una versión diferente en la “Staging Area” dicho archivo volverá a esa versión en el directorio de trabajo, de lo contrario devolverá el archivo a su versión anterior que se encuentra dentro del repositorio local. |
| git commit -m “mensaje” | Un “commit” es un registro versionado dentro de la base de datos de Git en nuestro repositorio local, por tanto, este comando se utiliza para enviar los archivos que se encuentran en la “Staging Area” a nuestro repositorio local. Es buena práctica y obligatorio añadir un mensaje (-m) para poder señalar los cambios realizados, ya que les “commit” suelen tener códigos extensos como nombre, los que serían imposibles de reconocer a que cambios realizados pertenecen.  Cabe destacar que estos registros de versiones van acompañados del nombre y correo de quien los ha realizado, |
| git diff “Nombre del archivo” | Compara la versiones modificas de un archivo presente el directorio de trabajo con el presente en la “Staging Area”, si dentro de esta área no se encuentran versiones modificadas del archivo, la comparará con la última versión presente en el repositorio local. |
| git diff | Compara la versiones modificas de todos los archivos presente el directorio de trabajo con las presentes en la “Staging Area”, si dentro de esta área no se encuentran versiones modificadas de los archivos, las comparará con las últimas versiones presentes de cada archivo en el repositorio local. |
| git diff “código de commit” “código de commit diferente” | Compara los cambios realizados en los archivos, entre las versiones de “commit” indicadas. Cabe destacar que se respeta la línea temporal, por ende, el orden de los códigos expuestos en el comando afectará la forma en que se ve la información. |
| git log | Lista el historial del “commit” con sus respectivos códigos, información de autoría y mensaje. |
| git show | Muestra los cambios realizados a los archivos en el último “commit” enviado. Si hubo archivos que no sufrieron modificaciones previo al envío del “commit” no se mostrará información de dicho archivo. |
| git show “Nombre del archivo” | Muestra los cambios realizados a un archivo concreto en el último “commit” enviado. |
| git checkout “código del commit” | Permite ver un espacio de trabajo con versiones de archivos perteneciente a ese log, se puede trabajar con los archivos, modificarlos a partir de esas versiones.  Se debe tener cuidado de no hacer “commit” porque el log HEAD cambiaría a esta versión anterior. Otra opción es crear una rama nueva con esta versión de dichos archivos mediante el comando git switch -c “Nombre de la nueva rama”. Para volver a la versión actual del espacio de trabajo, ejecutar el comando “git checkout nombre de la rama”. |
| git checkout “código del commit” “nombre del archivo” | Permite ver y trabajar en el espacio de trabajo la versión anterior de un archivo específico correspondiente a ese “commit” en particular, se debe procurar no hacer “commit” ya que se guardará dicha versión en el head. Para volver a la versión actual del espacio de trabajo, ejecutar el comando “git checkout nombre de la rama nombre del archivo”. |
| git reset “código de commit” --hard | Vuelve los archivos del área de trabajo a una versión anterior, la correspondiente en repositorio local bajo el código del “commit” indicado. Los datos dispuestos en la “Staging Area” se eliminan. |
| git reset “código de commit” --soft | Elimina el historial de “commit” y versiones de los archivos hasta el indicado en el comando, dentro del repositorio local, pero conserva las versiones del espacio de trabajo y la “Staging Area”. |
| **Trabajo con ramas (Branches)** | |
| git branch | Lista las ramas disponibles. |
| git branch “nombre rama” | Permite crear una nueva rama. |
| git branch -m “nombre rama” | Permite modificar el nombre de la rama en que se está situado. |
| git checkout “nombre rama” | Cambia de rama. |
| git checkout -b “nombre rama” | Crea una rama con el nombre indicado y cambia automáticamente a dicha rama. |
| git branch -d “Nombre de la rama” | Elimina una rama que no haya realizado algún “commit”, se debe ejecutar el comando desde una rama diferente a la que se desea eliminar.  Si existen archivos en el espacio de trabajo o en la “Staging Area” estos quedarán en dichos lugares, independiente de la rama en la que nos encontremos, incluso una vez eliminada la rama en donde se crearon, permanecerán allí hasta que sean enviada al repositorio local desde alguna otra rama. Esto sucede porque las ramas solo se diferencian una vez realizado sus respectivos “commit”, espacio de trabajo y “Staging Area”, son compartidos. |
| git branch -D “Nombre de la rama” | Fuerza la eliminación de una rama que haya realizado algún “commit”, se debe ejecutar el comando desde una rama diferente a la que se desea eliminar.  Si existen archivos en el espacio de trabajo o en la “Staging Area” estos quedarán en dichos lugares, independiente de la rama en la que nos encontremos, en incluso una vez eliminada la rama en donde se crearon, permanecerán allí hasta que sean enviada al repositorio local desde alguna rama diferente. Esto sucede porque las ramas solo se diferencian una vez realizado sus respectivos “commit”, espacio de trabajo y “Staging Area”, son compartidos. |
| git merge “Nombre de la rama a fusionar” | Permite fusionar dos ramas, este comando debe ejecutarse desde la rama que recibirá los cambios. |
| **Interacción con repositorio remoto** | |
| git remote add origin “url del repositorio remoto” | Conecta el repositorio local a uno remoto, guarda en las configuraciones el lugar donde debe conectarse nuestro repositorio local para subir, empujar, los archivos a la nube. |
| git push origin “Nombre de la rama” | Envía los archivos a un repositorio remoto, y a la rama indicada. Es muy probable que pida credenciales para realizar la acción. |
| git pull origin “nombre de rama” | Actualiza los cambios almacenados en un repositorio remoto que no están presente en el repositorio local, en este caso de una rama especifica. |
| git push | Envía los cambios de todo el repositorio local al repositorio remoto. |
| git pull | Actualiza todos los archivos disponibles en el repositorio remoto que no se encuentre en el repositorio local. |