1. Comandos útiles de Unix:

- ls -la: lista archivos incluyendo ocultos, info de usuario, permisos, etc.
- cd path: cambia el directorio de trabajo actual.
- pwd: muestra el directorio de trabajo actual.
- whoami: muestra la identidad actual del usuario.
- mkdir path: crea un nuevo directorio.
- rmdir path: borra el directorio indicado.
- mv source target: mueve un fichero o carpeta a la ubicación indicada.
- rm path: elimina un archivo. (-rf, borrado recursivo y forzoso).
- rename oldname newname: cambia el nombre de un fichero.
- command > file: redirige la salida del comando al archivo indicado.
- first | second: la salida del primer comando se usa como entrada del segundo.
- less: muestra el contenido de un fichero. Basado en Vim, por tanto más potente que el comando cat.
- history: muestra el historial de comandos usados recientemente.
- clear: limpia la consola.
- bash filename: ejecuta los comandos guardados en un archivo.

2. Conceptos clave de Git:

Estadios en los que puede encontrarse nuestro código (nuestros cambios sobre el contenido de los ficheros, en realidad).

1. **Workspace**: Es el estado real de nuestros ficheros. Tal y como los vemos en nuestro editor.
2. **Stage**: Aquí se encuentran los cambios sobre nuestros ficheros que se incluirán en el próximo *commit*. Cuando hacemos un git add, un git rm o un git mv, estamos introduciendo cambios en el *stage*, indicándole a Git que en el próximo *commit* esos cambios irán incluidos.
3. **Commits** (locales): Cada *commit* es un grupo de cambios sobre uno o varios ficheros, con una descripción, una fecha, un autor, etc. La gran diferencia con *SVN* es que los commits en *Git* son locales hasta que no

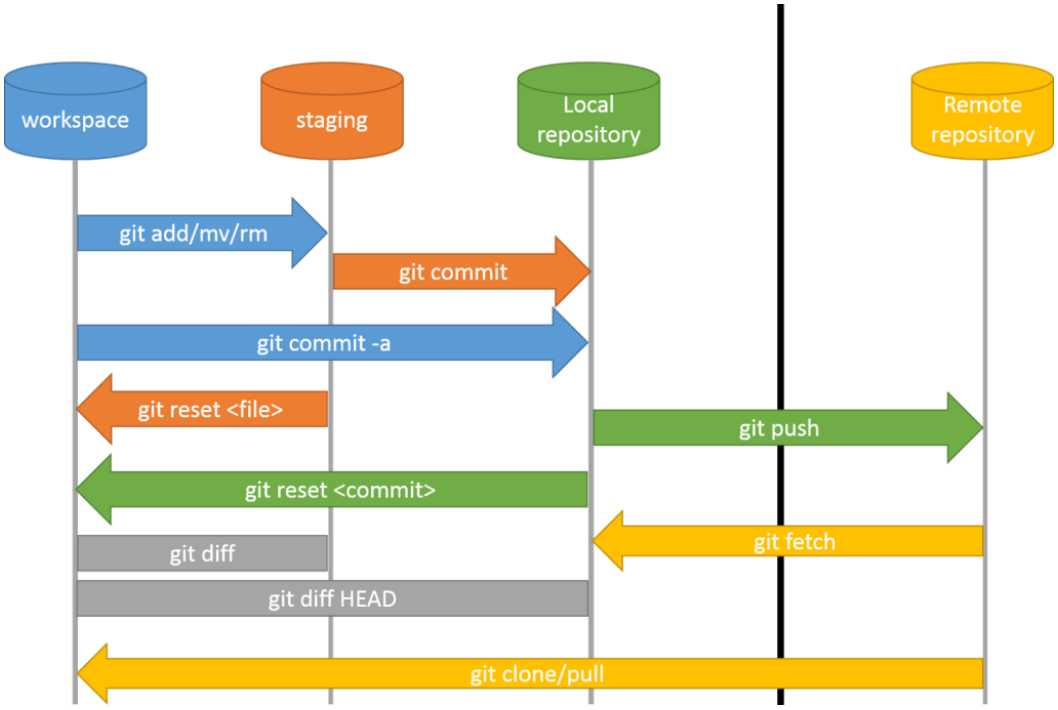
se efectúa la subida al servidor. Estos commits locales (**importante que sean locales**) pueden ser modificados sin peligro (con *modificados* quiero decir que se les pueden añadir más cambios, actualizar su mensaje o incluso eliminarlos).

4. **Commits** (remotos): Cuando se suben cambios al servidor (o como se le llama en Git: el *remoto*), se considera que estos entran a formar parte del histórico compartido entre los desarrolladores del proyecto y, por lo tanto, no es buena práctica modificarlos del mismo modo en que se hace cuando los *commits* son locales (además hacerlo puede provocar importantes quebraderos de cabeza).

En resumen, el *flow* de trabajo con Git es:

1. Hago cambios en mis ficheros (**workspace**).
2. Añado al **stage** los cambios que quiero commitear.
3. Hago el **commit.**
4. Subo los cambios al remoto.

3. Estructura y workflow de Git:



4. Primeros pasos con Git:

- Existen versiones de Git para Linux, macOs y Windows.
- La descarga e instalación en los distintos S.O. es muy sencilla:

  a. Instalación en Linux: https://git-scm.com/download/linux
  b. Instalación en macOS: https://git-scm.com/download/mac
  c. Instalación en Windows: https://git-scm.com/download/win

- Trabajo con línea de comandos:
  - Es fundamental aprender los conceptos de Git usando la línea de comandos. Después se podrá trasladar fácilmente estos conceptos al cliente gráfico que queramos usar.
  - En macOs o en Linux puedes usar el terminal. En Windows puedes usar Git Bash, PowerShell o Cmder.
- Una vez instalado Git, debes configurar tu nombre de usuario y tu email. Todos los commits que hagas llevarán esta información:
  - git config --global user.name "Antonio Pérez"
  - git config --global user.email antonio.perez@gmail.com
- Puedes configurar muchos otros elementos, como el editor que se usará cuando Git necesite que introduzcas un texto:
  - git config --global core.editor nano
- Puedes comprobar los parámetros configurados con el comando:
  - git config --list

5. Comandos de Git:

- git init: crear un nuevo repo local en la carpeta sobre el que se ejecute. Para eliminar repositorio creado con git init usaremos rm -rf .git.
- git clone: clona un repo existente, local o en la nube.
- git status: informa de que ficheros tenemos staged, listos para ser comiteados y que archivos tenemos pendientes del git add.
- git add: pasa a archivos a staged, preparados para hacer commit.
- git rm: Se usa para eliminar archivos de un repositorio de Git. Agrega los cambios al área de stagging. Permite no solo eliminar un archivo del repositorio, sino también, si lo desea, del sistema de archivos.

- git mv: mueve o renombra un archivo. Agrega los cambios al área de stagging.
- git restore --staged <file>: elimina el archivo del staging area, dejando sus modificaciones reales intactas en el working directory. Lo usaremos como comando opuesto a git add <file>.
- git restore <file>: si un fichero ha sido eliminado del working directory, podemos restarurarlo con esta opción, tanto desde el área de staging como desde el repositorio local.
- git restore --source <commit hash> <file>: restaura el fichero del working directory a su estado en el commit indicado.
- git commit: comitea al working directory cambios del staged area.
    - -m: para incluir un mensaje en el commit.
    - -a: comitea al working directory cambios del workspace. Solo funciona con aquellos que se han agregado con git add en algún momento de su historial.
    - --amend: permite modificar el commit más reciente, por ejemplo cambiando el mensaje del commit o añandiéndole algún cambio staged del que nos hayamos olvidado.
- git log: muestra el historial de commits del repositorio.
    - --oneline: muestra los commits de manera más compacta.
    - -numCommits: muestra el número de commits indicado mediante numCommits, ej:-3.
    - -p: muestra autoría, fecha y cambios en cada commit.
    - --pretty: modifica el formato de la salida y dispone a su vez de varias opciones, una especialmente útil es format.
        - git log --pretty=format:"%h - %an, %ar : %s"
- git show <commit hash>: muestra los cambios concretos en el códgio del commit indicado.
- git diff: diferencias entre el local repository y el working directory.
- git diff --staged: diferencias entre el local repository y el staging area.
- git diff HEAD HEAD ~1: diferencias entre las dos últimas versiones.
- git difftool: compara cambios en archivos. En ocasiones es más práctico emplear un GUI. Por ejemplo abriendo los archivos con VS

Code, se pueden ver cambios entre revisiones. Puedes configurar tu difftool preferida con el comando: git config –global -e.

- git reset: elimina commits del repositorio local.
    - <commit hash>: situa el HEAD del local repository en el commit indicado eliminando los posteriores. Además, pasa al workspace los ficheros modifcados en los commits eliminados.
    - --soft HEAD~1: situa el HEAD del local repository en el commit indicado eliminando los posteriores. Además, pasa al staging area los ficheros modifcados en los commits eliminados.
    - --hard HEAD~1: es necesario utilizar este comando con mucho cuidado, ya que elimina el último commit y además se pierden los cambios correspondientes a dicho commit, incluso machacando los ficheros de nuestro working directory con su contenido de la versión anterior.
- git revert <commit hash>: deshace el último commit. Añade un nuevo commit, que comienza con la palabra “revert” y deshace los cambios introducidos por el último commit. Si ese commit incluía cambios en varios ficheros, podremos elegir que cambios revertir y cuales mantener. Este comando es más seguro que reset.
- git revert HEAD…HEAD~2: revierte todos los cambios en el rango.
- git blame <file>: muestra los autores de cambios en el fichero indicado.
    - -L numLinea: muestra los autores de cambios en la línea indicada. Se usa juntamente con <file>.
- git remote: muestra los repositorios remotos a los que tenemos conectados nuestro proyecto local.
    - -v: muestra las URLs asociadas que serán usadas al leer y escribir en ese remoto.
- git remote add <branch name> <url>: agrega el repositorio remoto indicado.
- git remote rename <old> <new>: **renombra una conexión remota.**
- git remote rm: elimina una URL remota de tu repositorio. no elimina el repositorio remoto del servidor. Simplemente, elimina de tu repositorio local el remoto y sus referencias.

- git pull <remote> <branch name>: baja contenido de un repositorio remoto. Si ese repositorio no existe todavía en nuestro local, será necesario hacer antes un git init. Si el proyecto ya existía en nuestro local y existen cambios, éstos se descargarán y se fusionarán con nuestros cambios locales automáticamente. Tiene el mismo efecto que un git fetch seguido de un git merge.
- git fetch <remote>: comprueba si existen novedades en un remoto y las descarga. A diferencia del pull, no las lleva directamente al working directory, machacando lo que allí pudieramos tener, sino que deja los cambios en una nueva rama. De esta manera, podremos hacer un merge y decidir que cambios nos interesa llevarnos al working directory y cuales no.
- git push <remote> <branch name>: envía cambios del local repository al remote repository.
- git branch: lista las ramas locales.
    - <nueva rama> <rama de referencia>: crea una rama.
    - -a: listas todas las ramas existentes, incluyendo las remotass.
    - -d <branch name>: elimina la rama indicada.
- git checkout <branch>: nos sitúa en la rama indicada.
- git checkout <commit hash>: crea una rama temporal con el HEAD apuntando al commit que hemos indicado. Puede ser útil puntualmente para ver el estado de un proyecto en un commit pasado. La rama temporal desaparece al hacer de nuevo git checkout master.
- git merge <branch name>: Mergea la rama indicada con la actual.
- git tag: Lista por orden alfabético todas las etiquetas creadas.
    - -a: crea un tag de tipo anotado.
    - -m: especifica el mensaje de la etiqueta, como en los commit.
        - Ejemplo: git tag -a v1.0 -m ’Version 1.0 de la aplicación’
- git show <tag>: muestra la información del tag indicado.