**SETUP Configuring user information used across all local repositories**

git version

show git version

git config --global user.name “[firstname lastname]”

**set a name that is identifiable for credit when review version history**

git config --global user.email “[valid-email]”

**set an email address that will be associated with each history marker**

git config --global color.ui auto

**set automatic command line coloring for Git for easy reviewing**

git config --global --list

**show current user details**

git config --global --unset-all user.name

**unset current user**

**SETUP & INIT**

git init

**initialize an existing directory as a Git repository**

git clone [url]

**retrieve an entire repository from a hosted location via URL**

**STAGE & SNAPSHOT**

git status

**show modified files in working directory, staged for your next commit**

git add [file]

**add a file as it looks now to your next commit (stage)**

git reset [file]

**unstage a file while retaining the changes in working directory**

git diff

**diff of what is changed but not staged**

git diff –staged

**diff of what is staged but not yet commited**

git commit -m “[descriptive message]”

**commit your staged content as a new commit snapshot**

## Temporarily switch to a different commit

$ git log

If you want to temporarily go back to it, fool around, then come back to where you are, all you have to do is check out the desired commit:

# This will detach your HEAD, that is, leave you with no branch checked out:

git checkout 0d1d7fc32

Or if you want to make commits while you're there, go ahead and make a new branch while you're at it:

git checkout -b old-state 0d1d7fc32

## Hard delete unpublished commits

If, on the other hand, you want to really get rid of everything you've done since then, there are two possibilities. One, if you haven't published any of these commits, simply reset:

# This will destroy any local modifications.

# Don't do it if you have uncommitted work you want to keep.

git reset --hard 0d1d7fc32

# Alternatively, if there's work to keep:

git stash

git reset --hard 0d1d7fc32

git stash pop

# This saves the modifications, then reapplies that patch after resetting.

# You could get merge conflicts, if you've modified things which were

# changed since the commit you reset to.

git revert --no-commit 0766c053..HEAD

git commit

**delete untrack files**

Step 1 is to show what will be deleted by using the -n option:

git clean -n

Clean Step - **beware: this will delete files**:

git clean -f

* To remove directories, run git clean -f -d or git clean -fd
* To remove ignored files, run git clean -f -X or git clean -fX
* To remove ignored and non-ignored files, run git clean -f -x or git clean -fx

**Note** the case difference on the X for the two latter commands.

If clean.requireForce is set to "true" (the default) in your configuration, one needs to specify -fotherwise nothing will actually happen.

Again see the [git-clean](http://git-scm.com/docs/git-clean) docs for more information.

Options

**-f**

**--force**

If the Git configuration variable clean.requireForce is not set to false, git clean will refuse to run unless given -f, -n or -i.

**-x**

Don’t use the standard ignore rules read from .gitignore (per directory) and $GIT\_DIR/info/exclude, but do still use the ignore rules given with -e options. This allows removing all untracked files, including build products. This can be used (possibly in conjunction with git reset) to create a pristine working directory to test a clean build.

**-X**

Remove only files ignored by Git. This may be useful to rebuild everything from scratch, but keep manually created files.

**-n**

**--dry-run**

Don’t actually remove anything, just show what would be done.

**-d**

Remove untracked directories in addition to untracked files. If an untracked directory is managed by a different Git repository, it is not removed by default. Use -f option twice if you really want to remove such a directory.

**BRANCH & MERGE**

**Isolating work in branches, changing context, and integrating changes**

git branch

**list your branches. a \* will appear next to the currently active branch**

git branch [branch-name]

**create a new branch at the current commit**

git checkout

**switch to another branch and check it out into your working directory**

git merge [branch]

**merge the specified branch’s history into the current one**

git log

show all commits in the current branch’s history

**INSPECT & COMPARE**

Examining logs, diffs and object information

git log

**show the commit history for the currently active branch**

git log branchB..branchA

**show the commits on branchA that are not on branchB**

git log --follow [file]

**show the commits that changed file, even across renames**

git diff branchB...branchA

**show the diff of what is in branchA that is not in branchB**

git show [SHA]

**show any object in Git in human-readable format**

**TRACKING PATH CHANGES**

Versioning file removes and path changes

git rm [file]

**delete the file from project and stage the removal for commit**

git mv [existing-path] [new-path]

**change an existing file path and stage the move**

git log --stat -M

**show all commit logs with indication of any paths that moved**

**SHARE & UPDATE**

**Retrieving updates from another repository and updating local repos**

git remote add [alias] [url]

**add a git URL as an alias**

git fetch [alias]

**fetch down all the branches from that Git remote**

git merge [alias]/[branch]

**merge a remote branch into your current branch to bring it up to date**

git push [alias] [branch]

**Transmit local branch commits to the remote repository branch**

git pull

**fetch and merge any commits from the tracking remote branch**

**REWRITE HISTORY**

**Rewriting branches, updating commits and clearing history**

git rebase [branch]

**apply any commits of current branch ahead of specified one**

git reset --hard [commit]

**clear staging area, rewrite working tree from specified commit**

**TEMPORARY COMMITS**

**Temporarily store modified, tracked files in order to change branches**

git stash

**Save modified and staged changes**

git stash list

**list stack-order of stashed file changes**

git stash pop

**write working from top of stash stack**

git stash drop

**discard the changes from top of stash stack**