**Git stash:**

* To save changes made when they’re not in a state to commit them to a repository. This will store the work and give a clean working directory. For instance, when working on a new feature that’s not complete, but an urgent bug needs attention.

**$ git stash –u**

* Stores current work with untracked files

**$ git stash pop**

* Brings stashed work back to working directory

**Git stash save**

* The modified files from working area and staging area will be kept in another area. That area is called stash area and makes them clean

**Git stash list**

* We can get the commit id, we can’t see them physically

**Git stash apply (to apply the changes)**

* But it won’t remove the stash after applying

**Git stash pop**

* It applies and removes the entry from stash
* We can do stashing from any branch

**Git stash save ‘message’**

* We can give a message to stash. In stash list, 0 is the first and 1 is before to that, then 2 is before to the 1 etc… 0 is the latest one

**Git stash show stash@{0}**

* It is to see the files in the stashing area

**Git stash apply stash@{0}**

* To apply the particular stash

**Git stash pop stash@{0}**

* To pop particular stash

**Git stash drop**

**Git stash drop stash@{0}**

* To drop the stash files

**Git stash clear**

* To remove all stash
* stash@{0} will be the latest, if we apply or drop or pop. It will do it only for the latest one. Rest will be ignored
* we can apply stash in any branch

**Git ignore file:**

* In. gitignore, we can mention whatever the files we want to ignore. So that git won’t allow to add or push them
* If we rename the file into. gitdontignore then it won’t be applied. Below is the example



* While creating a repository, there will be an option to add .gitignore file. By using that we can add
* We can select pre-configured files while creating a .gitignore file like java, maven etc…

**Git clean:**

* **git-clean** - Remove untracked files from the working tree

**-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.
* **git clean -n**
* **git clean -n -fd**
* is to show what will be deleted by using the -n option
* **git clean -f**
* this will delete files
* 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

**-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**

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

**git clean -i**

* That stands for interactive and you will get a quick overview of what is going to be deleted offering you the possibility to include/exclude the affected files. Overall, still faster than running the mandatory --dry-run before the real cleaning

**-n**

**--dry-run**

* Don’t actually remove anything, just show what would be done
* If untracked directory is a git repository of its own (e.g. submodule), you need to use -f twice:

**git clean -d -f -f**

* To remove all untracked files, The simple way is to add all of them first and reset the repo as below

**git add –all**

**git reset --hard HEAD**

**we need to commit the changes even after deleting the files in git local repo**

**git show:**

it is used to see the recent commits

**git show HEAD**

**git show HEAD~1**

**git show <commit ID> 🡪to see the changes that are made with this commit ID**