

01 Git configuration

<code>git config --global user.name "Your Name"</code>	Set the name that will be attached to your commits and tags.
<code>git config --global user.email "you@example.com"</code>	Set the e-mail address that will be attached to your commits and tags.
<code>git config --global color.ui auto</code>	Enable some colorization of Git output.

02 Starting a project

<code>git init [project name]</code>	Create a new local repository in the current directory. If [project name] is provided, Git will create a new directory named [project name] and will initialize a repository inside it.
<code>git clone <project url></code>	Downloads a project with the entire history from the remote repository.

03 Day-to-day work

<code>git status</code>	Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.
<code>git add [file]</code>	Add a file to the staging area . Use in place of the full file path to add all changed files from the current directory down into the directory tree .
<code>git diff [file]</code>	Show changes between working directory and staging area .
<code>git diff --staged [file]</code>	Shows any changes between the staging area and the repository .
<code>git checkout -- [file]</code>	Discard changes in working directory . This operation is unrecoverable .
<code>git reset [<path>...]</code>	Revert some paths in the index (or the whole index) to their state in HEAD .
<code>git commit</code>	Create a new commit from changes added to the staging area . The commit must have a message!

<code>git rm [file]</code>	Remove file from working directory and staging area .
----------------------------	---

04 Storing your work

<code>git stash</code>	Put current changes in your working directory into stash for later use.
<code>git stash pop</code>	Apply stored stash content into working directory , and clear stash .
<code>git stash drop</code>	Delete a specific stash from all your previous stashes .

05 Git branching model

<code>git branch [-a]</code>	List all local branches in repository. With -a : show all branches (with remote).
<code>git branch [branch_name]</code>	Create new branch, referencing the current HEAD .
<code>git rebase [branch_name]</code>	Apply commits of the current working branch and apply them to the HEAD of [branch] to make the history of your branch more linear.
<code>git checkout [-b] [branch_name]</code>	Switch working directory to the specified branch. With -b : Git will create the specified branch if it does not exist.
<code>git merge [branch_name]</code>	Join specified [branch_name] branch into your current branch (the one you are on currently).
<code>git branch -d [branch_name]</code>	Remove selected branch, if it is already merged into any other. -D instead of -d forces deletion.

Commit	a state of the code base
Branch	a reference to a commit; can have a tracked upstream
Tag	a reference (standard) or an object (annotated)
HEAD	a place where your working directory is now

06 Inspect history

<code>git log [-n count]</code>	List commit history of current branch. -n count limits list to last n commits.
<code>git log --oneline --graph --decorate</code>	An overview with reference labels and history graph. One commit per line.
<code>git log ref..</code>	List commits that are present on the current branch and not merged into ref . A ref can be a branch name or a tag name.
<code>git log ..ref</code>	List commit that are present on ref and not merged into current branch.
<code>git reflog</code>	List operations (e.g. checkouts or commits) made on local repository.

07 Tagging commits

<code>git tag</code>	List all tags.
<code>git tag [name] [commit sha]</code>	Create a tag reference named name for current commit. Add commit sha to tag a specific commit instead of current one.
<code>git tag -a [name] [commit sha]</code>	Create a tag object named name for current commit.
<code>git tag -d [name]</code>	Remove a tag from local repository.

08 Reverting changes

<code>git reset [--hard] [target reference]</code>	Switches the current branch to the target reference , leaving a difference as an uncommitted change. When --hard is used, all changes are discarded. It's easy to lose uncommitted changes with --hard .
<code>git revert [commit sha]</code>	Create a new commit, reverting changes from the specified commit. It generates an inversion of changes.

09 Synchronizing repositories

<code>git fetch [remote]</code>	Fetch changes from the remote , but not update tracking branches.
<code>git fetch --prune [remote]</code>	Delete remote Refs that were removed from the remote repository.
<code>git pull [remote]</code>	Fetch changes from the remote and merge current branch with its upstream.
<code>git push [--tags] [remote]</code>	Push local changes to the remote . Use --tags to push tags.
<code>git push -u [remote] [branch]</code>	Push local branch to remote repository. Set its copy as an upstream.

10 Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/Ubuntu please type in the terminal:

```
sudo apt-get install git
```

If you need to install Git from source, you can get it from git-scm.com/downloads.

An excellent Git course can be found in the great Pro Git book by Scott Chacon and Ben Straub. The book is available online for free at git-scm.com/book.

11 Ignoring files

```
cat <<EOF > .gitignore
```

```
/logs/*
```

```
!logs/.gitkeep
```

```
/tmp
```

```
*.swp
```

```
EOF
```

To ignore files, create a `.gitignore` file in your repository with a line for each pattern. File ignoring will work for the current and sub directories where `.gitignore` file is placed. In this example, all files are ignored in the logs directory (excluding the `.gitkeep` file), whole tmp directory and all files `*.swp`.