# Introduction

## Version Control

Version control is a concept that lets developers manage the evolution of files in a project (addition, modification, or deletion). Each of these versions (revisions) is saved on the a server (Git repository) so that it is possible to go back to them at any point in time and start again from there.

A modification therefore constitutes the evolution between two versions. We can therefore speak of the difference between two versions as well as of modifications leading to a new version.

To be able to make changes to the repository, it should be copied (forked or cloned) locally. Then, the developer makes modifications and tests locally, then submit the changes to update the repository - this is called a “commit”.

Version control is not only possible with a single developer, but a whole team can clone and commit their changes on the repository. If each developer works on separate files or separate parts of a file, then commit their changes, both versions will be merged automatically. Otherwise, conflicts can arise, which must be resolved (merged) manually for the commit to be accepted on the repository.

When divergent modifications occur without conflict, branches are created, which can be merged back into one. Branches are used for the maintenance of old versions of the software on a separate branch, while continuing the development of future versions on the main branch. This is useful if some users want to keep using an old major version of the project (1.x) instead of switching to the newer one (2.0) because of breaking changes; the 1.*x* version still has some errors, which will be fixed with 1.*x*.*y* versions, even after the 2.0 version has been released.

## Git

Git is a command-line tool for version control of projects - mostly used for programming.

To start working on a project with Git, the developer has to fork or clone it to start working on it locally. and then has to to propose its contribution (pull request) to the repository to be able to commit.

Git has the ability to mark (tag) a certain commit with a name, used mostly for important milestones, like major (*x*.0.0), minor (*x*.*y*.0), or patch (*x*.*y*.*z*) versions, or named versions.

## GitHub

GitHub is a web service for management of Git repositories.

Among other things, GitHub offers services such as version and tag management, continuous integration, and documentation for each project in Markdown (.md).

GitHub Desktop is a tool for MacOS and Windows which is an alternative for the Git command line. It makes it easier to clone, fork, and upload a repository, see changes in files since the last commit (diffs), add comments before a commit, undo commits, manage branches, and more.

# Using Git

## Download

Git can be downloaded for Windows, MacOS, and Linux from the [Git Downloads Page](https://git-scm.com/downloads).

## Usage

The general usage of the git command is as follows:

Each option and commands are described in the [Git Documentation](https://git-scm.com/docs/git" \l "_options).

|  |  |
| --- | --- |
| $ | git [--version] [--help] [-C <*path*>] [-c <*name*>=<*value*>]  [--exec-path[=<*path*>]] [--html-path] [--man-path] [--info-path]  [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]  [--git-dir=<*path*>] [--work-tree=<*path*>] [--namespace=<*name*>]  [--super-prefix=<*path*>] [--config-env=<*name*>=<*envvar*>]  <command> [<*args*>] |

## Configuration

After installation, the user information should be set so that it can be attached to commits:

|  |  |
| --- | --- |
| $  $ | git config --global user.name "<*name*>"  git config --global user.email "<*email*>" |

### Working with Git

### Create or Clone Repository

To create a Git repository from a local directory, use git init. To add it to an existing empty GitHub repository, use git remote add origin <*url*>.

To clone a Git repository to a local directory, use git clone <*url*>.

### Synchronize with Repository

To download the repository history, use git fetch.

To combine a branch with the current directory/branch, use git merge.

To update the current directory/branch with new commits from the repository, use git pull, which is equivalent to git fetch && git merge.

To update the repository with new commits from the current directory/branch, use git push.

### Commit on Branches

To add a file from the workspace to the commit queue (index), use git add "<*file name*>".

To commit the files in the index to the local repository/branch, use git commit -m "<*comment*>". The comment should be descriptive about the changes made in the commit.

Each commit has a particular name, which is a SHA hash, which has a long (full) and a short version. To get the hash of the last commit, use git show -s --format=%H for the long version of the hash, or git show -s --format=%H for the short version of the hash.

If a mistake has been made, to revert the commit, use git reset <*commit*>. This will revert changes from the repository only and local files will not be affected. To affect local files as well, use git reset --hard <*commit*>.

To show the difference between local changes and a commit, use git show <*commit*>.

To upload (push) all the waiting commits from the local repository to the upstream repository, use git push.

### Manage Branches

The commits are made on the local repository, on the branch in which the developer is checked out. To see the name of this branch, use git status.

To create a new branch, use git branch <*branch*>. The branch name should not contain spaces and special characters like ~, ^, \*, <, >, ?, /, \, |, :, ., [, and ".

To switch (checkout) to another branch, use git checkout <*branch*>. This will also update the local directory to contain the files from that branch.

To merge a branch into the currently checked out branch, use git merge <*branch*>.

To delete a branch that has been merged, use git branch -d <*branch*>. To force delete a branch without merging, use git branch -D <*branch*>.