Git

Alexander Solovyev

About

Git was created by Linus Torvalds, creator of Linux, in 2005 and was designed to do version control on Linux kernel

Version control

Version control systems are a category of software tools that help a lot of people manage changes to source code over time

Developers can review project history to find out:

- Who made changes in project?
- What changes were made?
- When were the changes made?

Git

Every software engineering project is an iterative process

Developer’s needs:

- understand what is happening in the project
- be able to work in an isolated environment
- restore the historical state of the project
- synchronise his work with the work of other developers

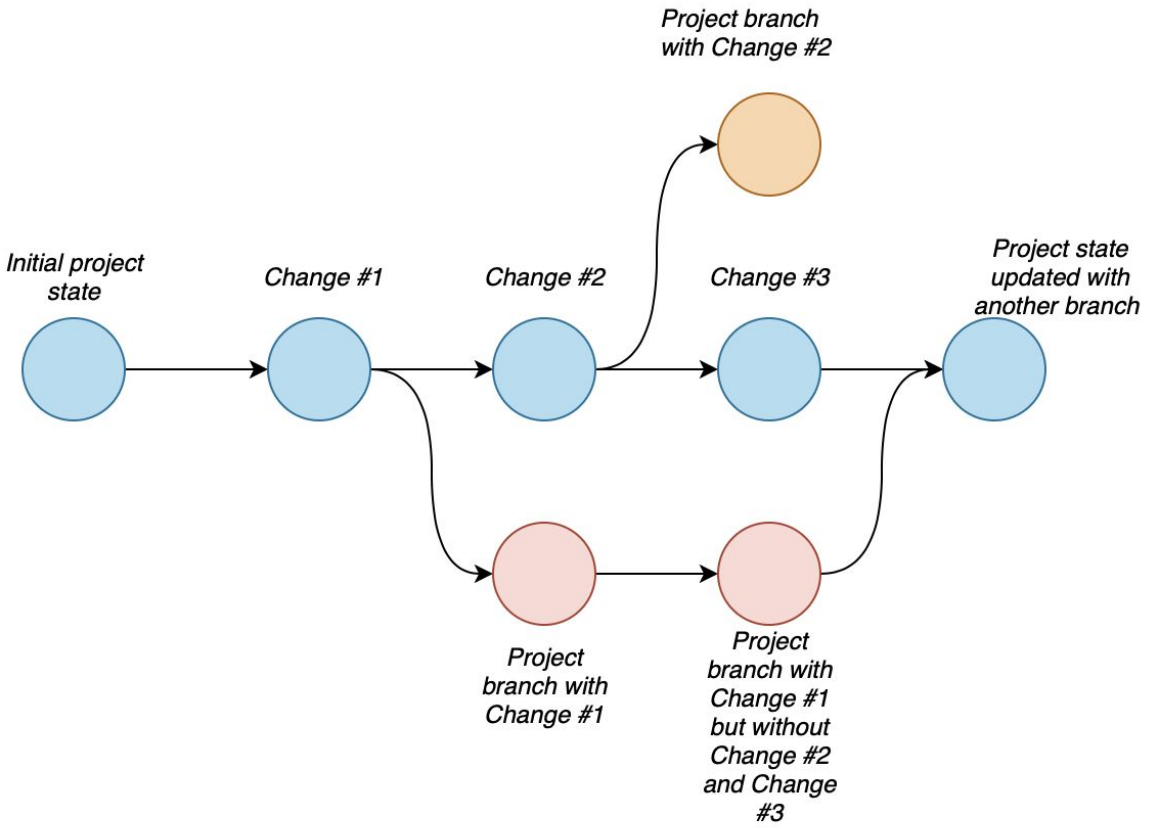
Git is an example of a *distributed version control system* (*DVCS*) commonly used for open source and commercial software development

DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes

Repository

A repository (Git project) contains all source files along with all branches of the project and all change data for that project

Commit - snapshot of a project in time



Configure

Configure your Git username and email using the following commands

```
$ git config --global user.name "<Enter Your Username>"
$ git config --global user.email "<your@email.com>"
```

You can call `git config –list` to verify these are set

Initialise new Git repository

To initialise new git repository you need to use command `git init`

Executing `git init` creates a *.git* subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository

This metadata includes subdirectories for *objects*, *refs*, and *template files*

A *HEAD* file points to the currently checked out commit

Aside from the *.git* directory, in the root directory of the project, an existing project remains unaltered

```
# Let's create new forlder for our git repo
$ mkdir project_repo
$ cd project_repo/
$ git init
# Or just git init <path to project directory>
$ cd ..
$ tree -a
.
└── project_repo
    └── .git
        ├── HEAD
        ├── branches
        ├── config
        ├── description
        ├── hooks
        ├── info
        ├── objects
        └── refs
```

Add remote

remote - is a remote git storage

To add this repository on your GitHub you need to have GitHub account

```
$ cd project_repo/
$ echo "# project_repo" >> README.md
# Add all files to prepearing commit
# add changes from all tracked and untracked files
$ git add -A .
# Create commit with this files
$ git commit -m "Commit message"
# On this step you should create project_repo repository in your GitHub
$ git remote add origin https://github.com/<your login>/project_repo.git
# Then it requires your GitHub login and password or use
# git remote add origin git@github.com:<your login>/project_repo.git

# Send your commits to GitHub in master branch
$ git push -u origin master
```

git clone

GitHub provides links to its repositories. When you created a repository on GitHub you can immediately clone it to your local environment

```
$ tree -a
.
0 directories, 0 files
$ git clone https://github.com/gvanrossum/pytype
Cloning into 'pytype'...
remote: Enumerating objects: 9545, done.
remote: Total 9545 (delta 0), reused 0 (delta 0), pack-reused 9545
Receiving objects: 100% (9545/9545), 10.90 MiB | 2.94 MiB/s, done.
Resolving deltas: 100% (7814/7814), done.
```

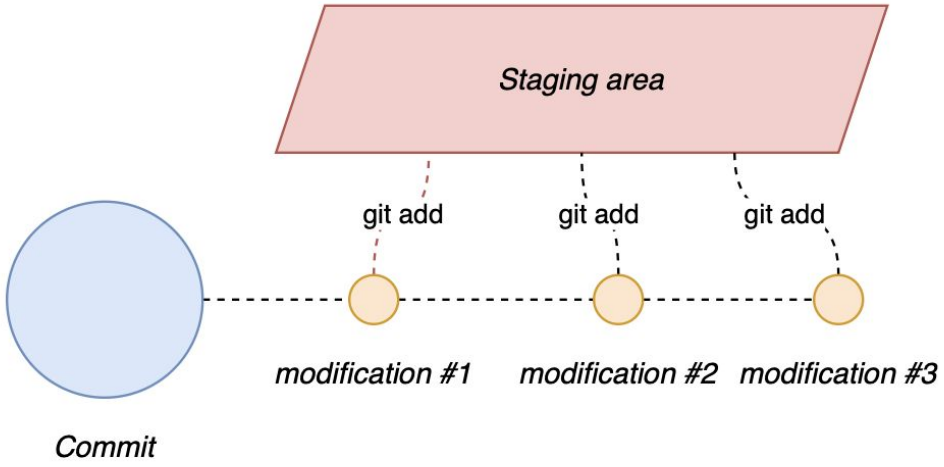
git clone --depth

You can use argument ***—depth*** to make shallow clone where only latest commits will be catched

```
$ git clone -depth=1 https://github.com/gvanrossum/cpython
Cloning into 'cpython'...
remote: Enumerating objects: 4808, done.
remote: Counting objects: 100% (4808/4808), done.
remote: Compressing objects: 100% (4372/4372), done.
remote: Total 4808 (delta 485), reused 1525 (delta 308), pack-reused 0
Receiving objects: 100% (4808/4808), 24.76 MiB | 2.58 MiB/s, done.
Resolving deltas: 100% (485/485), done.
Checking out files: 100% (4568/4568), done.

$ git clone https://github.com/gvanrossum/cpython
Cloning into 'cpython'...
remote: Enumerating objects: 770125, done.
remote: Total 770125 (delta 0), reused 0 (delta 0), pack-reused 770125
Receiving objects: 100% (770125/770125), 310.22 MiB | 3.60 MiB/s, done.
Resolving deltas: 100% (614400/614400), done.
Checking out files: 100% (4568/4568), done.

# 24.76 MiB vs. 310.22 MiB
# See the difference
```

git add

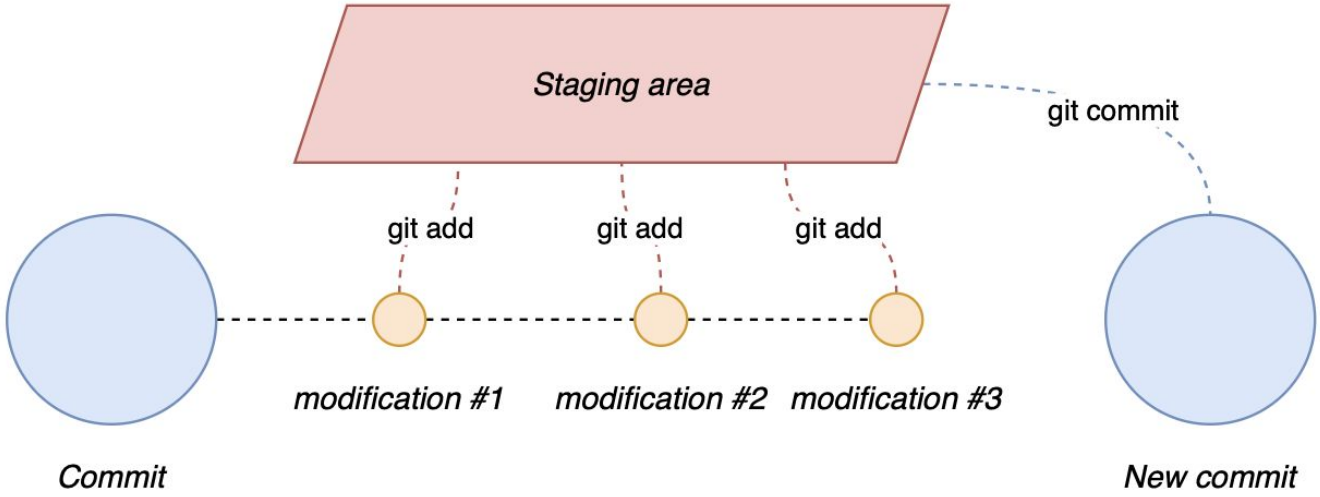
- `git add` command adds a change in the working directory to the staging area
- `git add` doesn't really affect the repository in any significant way — changes are not actually recorded until you run `git commit`
- ***Staging area*** is cumulative data buffer before final snapshot



git commit

Now that your staging area is set up the way you want it, you can commit your changes

Remember that anything that is still unstaged won't go into commit



git commit

```
# Commit the staged snapshot
$ git commit
# Commit a snapshot of all changes in the working directory
$ git commit -a
# A shortcut command that immediately creates a commit with a passed commit message
$ git commit -m "commit message"
# Immediately creates a commit of all the staged changes and takes an inline commit message
$ git commit -am "commit message"
```

Example

```
$ tree
.
├── README.md
└── example.py
0 directories, 2 files

# example.py wasn't staged yet
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        example.py
nothing added to commit but untracked files present (use "git add" to track)
$ git add example.py

# example.py staged but not commited yet
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
        new file:   example.py

# stage is commited now
$ git commit -m "Add example.py file"
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
```

git diff

By default `git diff` will show you any uncommitted changes since the last commit

If we add line *import this* in ***example.py*** file:

```
$ git diff
diff --git a/example.py b/example.py
index e69de29..c4da7d1 100644
--- a/example.py
+++ b/example.py
@@ -0,0 +1 @@
+import this
```

git log

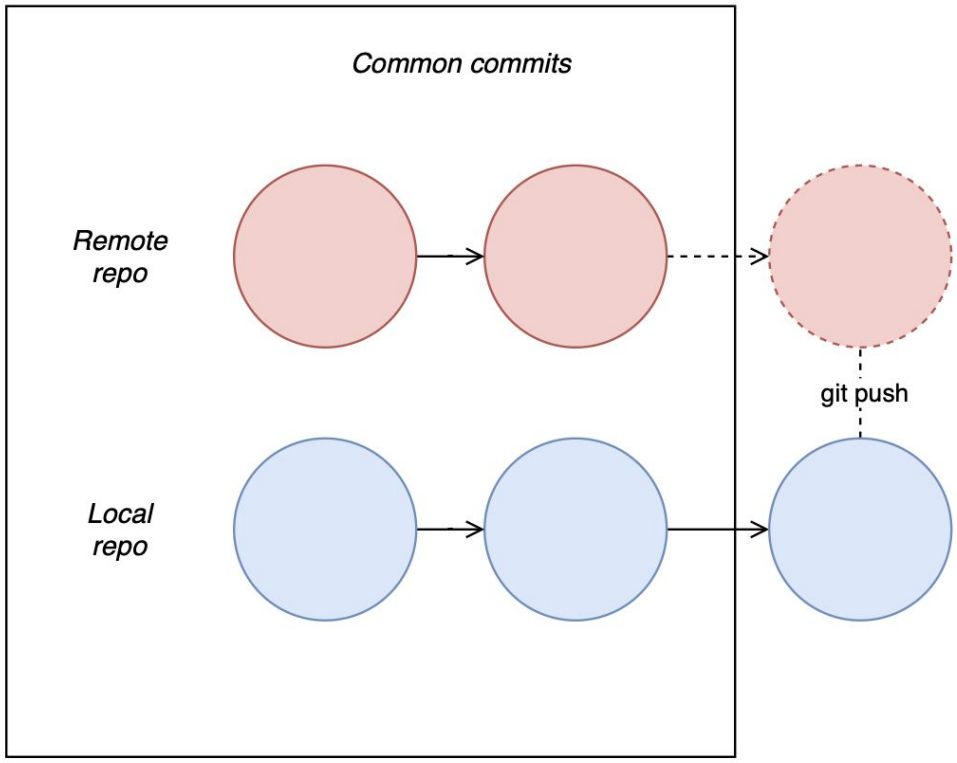
`git log` displays committed snapshots

```
$ git log
commit 5d3fad15a15e68a6c7b004294294873c540e29e4 (HEAD -> master)
Author: galeNightIn <nightingale.alex.info@gmail.com>
Date:   Wed Aug 5 01:42:32 2020 +0300

    Add example.py file

commit 341dd823714b311039ba5fc876e5e56f053b1321 (origin/master, origin/HEAD)
Author: galeNightIn <nightingale.alex.info@gmail.com>
Date:   Wed Aug 5 00:34:26 2020 +0300

    first commit
```

git push

git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo

```
git push <remote> <branch>
```



Example

Previously we committed our changes into local repo. It’s time to show our work to the world

```
# To see all remotes use this command. It's often only origin
$ git remote
origin
1. $ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 560 bytes | 560.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:galeNightIn/project_repo.git
   341dd82..386d1fd  master -> master
```

git fetch

The git fetch command downloads commits, files, and refs from a remote repository into your local repository

Fetching is what you do when you want to see or use other contributors work

- Get all branches from remote repo

```
git fetch <remote>
```

- Get specific branch from remote repo

```
git fetch <remote> <branch>
```

- Get all branches for all remotes

```
git fetch --all
```

git pull

The `git pull` command is used to fetch and download content from a remote repository and immediately update the local repository to match that content

```
git pull <remote>
```

git branch

A branch represents an independent line of development

Branches serve as an abstraction for the edit/stage/commit process

You can think of them as a way to request a brand new working directory, staging area, and project history

- List of branches: `git branch`
- Create new branch: `git branch <branch name>`
- Delete branch: `git branch -d <branch name> or git branch -D <branch name>`
- Rename branch: `git branch -m <new branch name>`

git checkout

The git checkout command lets you switch between the branches created by `git branch`

- Switch between existing branches

```
$ git branch
  dev
* master

$ git checkout dev

$ git branch
* dev
  master
```

- Create new branch based on HEAD and switch on it

```
$ git checkout -b new-branch
$ git branch
  dev
  master
* new-branch
```

- Create new branch based on and switch on it: `git checkout -b new-branch <branch>`