Git ABC

Kacper Kozerski

October 2024

1 TLDR

- Download the repository:

  ```
  git clone https://github.com/EnergySystemAnalysis-ETH/DeMuniRES.git
  ```

- Download the changes that others have made:

  ```
  git pull
  ```

- Create and jump to a branch that you will be working on:

  ```
  git checkout -b new_branch_name
  ```

- Stage all your changes in the current folder:

  ```
  git add .
  ```

- Create a commit out of staged changes (commit often!):

  ```
  git commit -m "message"
  ```

- Discard your changes, and return to the last commit:

  ```
  git reset --hard
  ```

- Push your local commits to the remote repo (read possible error messages!)

```
git push
```

- Do not push to main!
- If you want to merge your branch into the main - create a pull request using the GitHub web application

2 What is git?

2.1 Git

Git is a distributed version control system.

Distributed - developers can work on files stored directly on their computers (not on a remote server). However, the central repository is used to synchronize changes between the files that different users did

Version control - not only the most up-to-date versions of files are stored. Git allows one to look through the history of file changes, revert them, etc.

2.2 Repositories and their architecture

2.2.1 Repository

In the git system, the projects are stored in file repositories. There are two kinds of repositories:

- Remote repository - central server used to store data, enables us to share the code with other users, allows for settling conflicts when two users have changed the same file, etc. There is only a single remote repository per project
- Local repository - files stored on users' computers. There might be many local repositories representing the same project

2.2.2 Commit

The data in repositories is stored in the form of time snapshots, called "commits". Every time a developer makes some change to the code, they can stage their changes (choose which of them should be added to the repo) and add a commit, along with a message describing the change's content. One can then browse through the commits, check their contents, jump to them, etc.

2.2.3 Branches

To enable many developers to work in a parallel manner, without interrupting each other, branches were invented. If we were to imagine a series of commits as a timeline, branches are timelines that "branched out" from a different timeline at some point. The main branch is called "main" or "master" (the latter is not recommended anymore). The developer can then commit their changes to their branch, and not cause conflict with others. The branching timelines are then merged (after resolving conflicting changes) to reduce the number of branches (ideally to a single one - main).

2.3 Git and Github

At first central repos were created at the private servers. As it got more popular, different online hosting services were created to make this process easier. One of the most popular services of this kind is Github.

2.4 How to use git?

Git can be used:

- from the command line (git CLI), when one has to type in shell commands
- from the graphical interface of your favorite coding IDE. Possibly more comfortable for basic stuff, but impossible to do more complex things
- web application (for Github, Gitlab, etc.) - very limited functionality, but useful for things like pull requests, issue handling (specific to a hosting service)

The rest of the tutorial will assume the command line version, but IDE's graphical interfaces for git work in an analogous manner.

2.5 On AI usage

Some people use ChatGPT to help them with git problems. It might be helpful for beginners, but don't use it to resolve merge conflicts!

3 How to get data from the git repository?

3.1 Cloning into a repository

To create a local repository from a remote one - you have to clone into it:

```
git clone repo_link
```

3.2 Keeping repository up to date

To keep the repository up to date, you must pull (download) the changes made by different users. In the folder of your local repo:

```
git pull
```

If you have made some changes to your files, and pulling would overwrite them, the error will be raised, and the pull will not happen. To resolve this:

- If you want to save your changes:

  ```
  git stash
  ```

  Your changes will be kept in a stash, which will make them disappear for the time of the pull, but they can be easily retrieved with:

  ```
  git stash pop
  ```

- If you want to discard your changes:

  ```
  git reset --hard
  ```

This error should not happen in this project, since different students will be working on different features, and thus on different branches.

3.3 Browsing the git repository

Once in a git repository, you may look at the commit history of the branch you are at:

```
git log
```

You may also see the branches that are in the repo:

```
git branch
```

You may change your current version of files to a different branch:

```
git checkout branch_name
```

or to a specific commit:

```
git checkout commit_id
```

You can get the specific commit id from the git log.

4 How to work on a git repository?

4.1 Working on a branch

To create a new branch, either type:

```
git branch branch_name
```

or, to also jump to that branch after creation:

```
git checkout -b branch_name
```

The new branch will branch out from the branch you have been previously on. **Do not work on the main branch!**

4.2 How to commit your work?

Once you have made some changes to the code or data, you may check the status of your repository:

```
git status
```

The status will show which files have been changed from the ones from the last commit, which files are not tracked by git, etc.
If you want stage files or their changes you use:

```
git add filename1 filename2 folder1
```

Staged changes will be the content of the new commit.
If you want to add everything you have done you type:

```
git add .
```

The dot "." refers to the folder you are currently in (not a placeholder, you type in a literal dot).
To commit your changes:

```
git commit -m "change message"
```

Commit only happens in your local repository. **Commit often!** It's best to commit after every "atomic" change to your code, e.g. fixing a single bug, writing a single class, etc.
If you want to remove all staged changes you type:

```
git reset
```

This doesn't actually change the files themselves, only staged changes. If you want to discard all your changes and return to the newest commit, you type:

```
git reset --hard
```

4.3 How to push your data to a remote repository?

Your local commits should be pushed from time to time to the central repository - to allow others to access it, and to backup them. Once you have decided you want to push your commits, you type:

```
git push
```

The first push to a new branch might raise an error, but the error message usually tells you exactly what to do.

4.4 Merging the branches

If you want to merge the branches, you switch to a branch you want to merge into and type:

```
git merge source_branch_name
```

You may have to resolve possible conflicts, stage them, and create a commit. **Do not merge into the main!** Instead, create a pull request using the GitHub web application