Lecture 9: Git

Zhentao Shi and Zhan Gao

Jan 16, 2019

Git

Git is a version control system useful when developing and maintaining coding projects as well as preparing long documents.

Git helps me manage course materials, and I use Github https://github.com/zhentaoshi/econ5170 to share them. Free Git books and tutorials are available online.

- Atlassian Online Tutorial
- Udacity Course
- CodeAcademy Course
- Pro Git

We introduce some essential Git commands. There are two ways to interact with Git. Git provides a command line tool Git Bash, and there are many free Git GUIs available (I recommend SourceTree). Even if we use a GUI, knowing the basic commands is helpful.

There are lots of online tutorials about Git, for example this one. Self-learning is important.

Basic Commands

Local

- `git status` inspects the contents of the working directory and staging area.
- `git add filename` adds files to the staging area from working directory.
- `git add filename1 filename2` adds multiple files to the staging area.
- If the file is changed, we can use `git diff filename` to see the difference between the file in the working directory and the staging area. Reuse `git add filename` to add the updated file to the staging area.
- `git commit` stores changes from the staging area to the repository.
- `git commit -m "Commit Message"` The commit message must be in the quotation marks.
- `git log` displays historical commits stored chronologically in the repository.
- `git config --global user.name <name>`
- `git config --global user.email <email>`
- `git tag -a v1.0 -m 'message' [optional:commit-id]`
- `git rm --cached filename`
- `git branch [brach_name]`
- `git checkout [commit_id or branch name]`
- .gitignore

Eraser-like features

The latest commit is called **HEAD** commit * `git show HEAD` to view the HEAD commit. * `git checkout HEAD filename` to restore the file in the working directory to what you made in last

commit. * `git reset HEAD filename` this command unstages the file from committing in the staging area. This command resets the file in the staging area to be the same as the HEAD commit. It does not discard file changes from the working directory, it just removes them from the staging area. * `git reset SHA` This command works by using **the first 7 characters** of the SHA of a previous commit.

Branch

- `git branch` displays the current branch.
- `git branch brach_name` creates a new branch. The branch name cannot contain white-space
- `git checkout branch_name` switch to a branch
- `git merge branch_name` merging the branch into master **merge conflict**: Git doesn't know which changes you want to keep. Modify the file in the working directory and commit it again to avoid conflict.
- `git branch -d branch_name` deletes a branch from the project.

In Git, branches are usually a means to an end. We create them to work on a new project feature, but the ultimate goal is to merge that feature into the master branch.

Remote

- `git clone https://github.com/zhentaoshi/econ5170`
- `git remote -v` lists git project's remote copies.
- `git remote add origin`
- `git push origin master`
- `git pull`
- `git fetch` fetches the remote copy to the local hard disk. **Note:** This command will not merge changes from the remote into your local repository. It inspects the changes on the remote branch.

Collaboration typically works as follows: 1. Fetch and merge changes from the remote; 2. Create a branch to work on a new project feature; 3. Develop the feature on your branch and commit your work; 4. Fetch and merge from the remote again (in case new commits have been uploaded while you were working); 5. Push your branch up to the remote for review.

Conflict

Collaborators working on the same file separately might cause conflicts. In such situations, the command `git pull` will not merge chanegs from the remote into your local repository automatically due to the conflict. You are expected to see a similar message on the shell:

```
git pull
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://bitbucket.org/your-user-name/repo-name
   9b02654..0462f26  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

If you check the status by `git status`, you would be told that you have unmerged paths.

```
git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

To resolve the conflicts, we simply use a text editor, say *Sublime Text 3*, *Atom* or *Visual Studio Code*, to open the file with conflicts. In the illustrative example, it is the `README.md` file.

```
# Conflict-Resolving

<<<<<<< HEAD
Local changes.
=======
Remote changes.
>>>>>>> 0462f26ddfe83c35424168c2d7a0bed62c653413
```

You can see conflicts indicated by Git. The content between `<<<<<<< HEAD` and `=======` is on HEAD and content between `=======` and `>>>>>>>` is from the commit `0462f26ddfe83c35424168c2d7a0bed62c653413`. To resolve the conflicts, you just remove `<<<<<<< HEAD`, `=======` and `>>>>>> 0462f26ddfe83c35424168c2d7a0bed6` and keep what you want keep in the file.

In some cases, the difference between the local branch and the remote branch is significant, or the file with conflicts is of a special format, say Lyx file. It might be cumbersome to resolve all conflicts in the text editor. Another way to resovle conflicts is simply to open two versions of the files and copy and paste new updates and then commit the updated file. It easy to do so in *SourceTree*: select the commit -> right click the file -> click `Open Selected Version`.

In addition, we can resolve conflicts with *SourceTree* (**preferred**) or external merge tools, for example *KDiff3*. For details, it is recommended to refer to the answer on Stack Overflow.

Acknowledgment: Part of this notes is based on the course offered by CodeAcademy.