Git-Hub Repository

Git-Hub

Git is a version control system which means creating something (an application, for example), they are making constant changes to the code and releasing new versions, up to and after the first official (non-beta) release.

Version control systems keep these revisions straight, and store the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

Similarly, people who have nothing to do with the development of a project can still download the files and use them. Most Linux users should be familiar with this process, as using Git, Subversion, or some other similar method is pretty common for downloading needed files, especially in preparation for compiling a program from source code

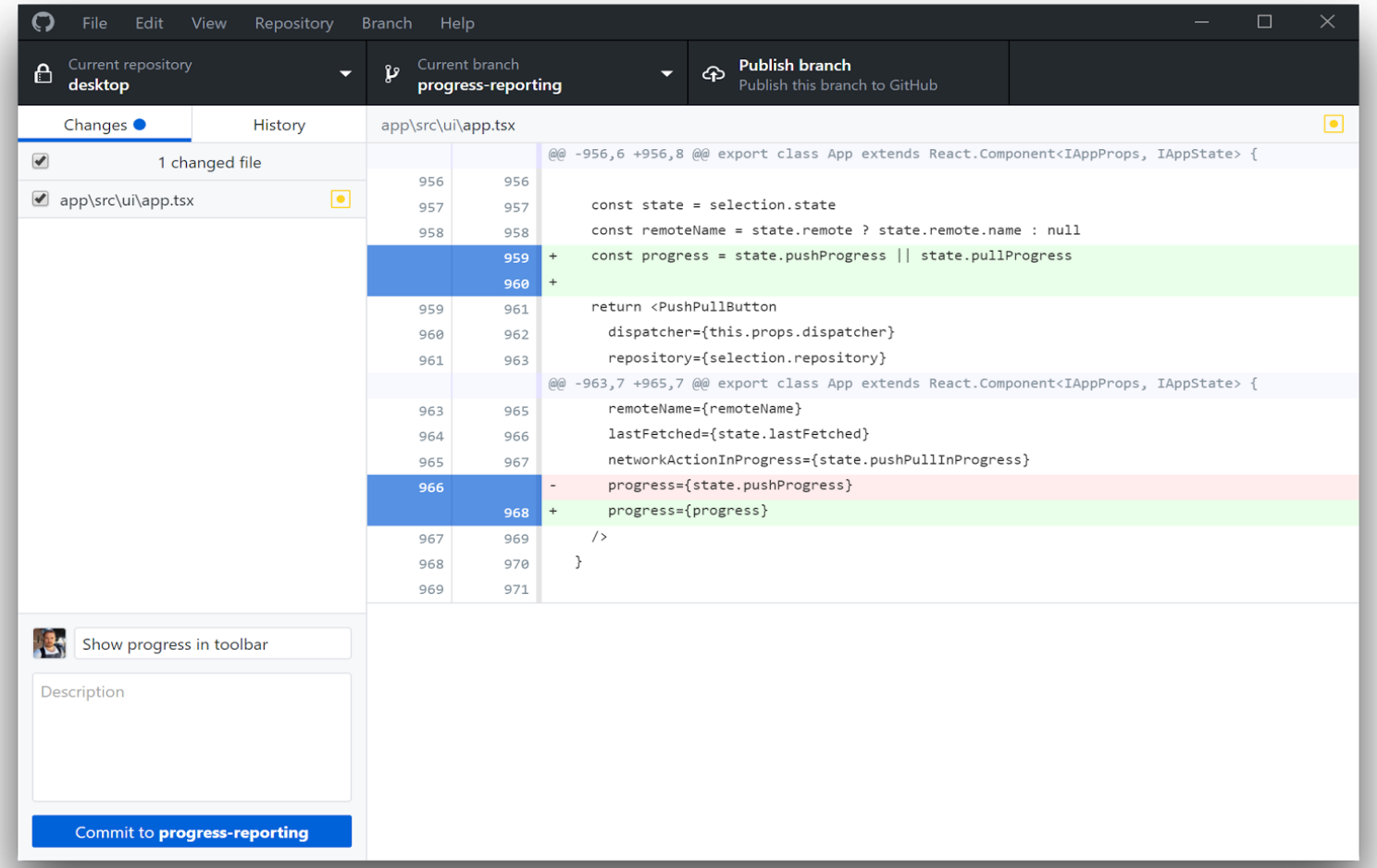
Repository

A repository is a location where all the files for a particular project are stored, usually abbreviated to “repo.” Each project will have its own repo, and can be accessed by a unique URL.

Git-Hub DeskTop

GitHub Desktop is an open source Electron-based GitHub app. It is written in TypeScript and uses React.The below image is reference for Git-Hub Desktop view.

Here we can select the multiple repositories,all the changes are displayed with name ina Changed tap.Finally commit the changes to master and Sync or publish the commits into repo.



How to Clone and Commit the codes into the Git-Repositories using Git-shell

Cloning the repo

By default, the command "git clone repository_name" will replicate the directory structure of your forked repository wherever the default Git Config tells it to.

If you want to clone the repository somewhere else, simply add the pathname at the end of the command

git clone repository_name /path/to/localrepo/

Adding files, Committing changes and Pushing

Whenever you want to update your online repository with your local changes, you need to follow these basic steps :

- **git add** all the new or modified files to your staging area
- Execute a **git commit** to take a snapshot of your local staging area
- **git push** the snapshot to your online repository

Whether you have modified an existing file or created a new file in the local repository, the procedure remains the same.
Simply type in
git add modified_filename1 modified_filename2 newly_created_filename1

This will add the specified files to your staging area.
If you do not call *git add* on a modified file, the staging area will simply keep the version of the file which existed when it was last added via *git add*.
You may also call
git add .

to add all modified or new files in your entire project to your staging area.
Once your staging area is ready, you must commit your changes by typing
git commit -m 'hopefully relevant message about this commit'

Finally, you push your commit to your online repository by typing :
git push remote_name branch_name
or more specifically :
git push origin development
The above command will push the changes from your currently active branch to your online repository's development branch.

Managing Branches

Making changes directly to your master branch is a bad idea. You should always have a working branch to try out your modifications on.
To list the available branches for your current project, type in :
git branch

To create a new branch, naming it whatever you want, type in :
git branch branch_name

To delete a branch, type in :
git branch -D branch_name

To switch to a branch, making it the currently active branch, type in :
git checkout branch_name

As an example, in order to return to your master branch, you would type in
git checkout master

Competing line change merge conflicts

To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.
For example, if you and another person both edited the file *styleguide.md* on the same lines in different branches of the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

1. Open Terminal.
2. Navigate into the local Git repository that has the merge conflict.
3. cd *REPOSITORY-NAME*
4. Generate a list of the files affected by the merge conflict. In this example, the file *styleguide.md* has a merge conflict.
5. git status
   ```
   # On branch branch-b
   # You have unmerged paths.
   #   (fix conflicts and run "git commit")
   #
   # Unmerged paths:
   #   (use "git add ..." to mark resolution)
   #
   # both modified:      styleguide.md
   #
   no changes added to commit (use "git add" and/or "git commit -a")
   ```
6. Open your favorite text editor, such as Atom, and navigate to the file that has merge conflicts.
7. To see the beginning of the merge conflict in your file, search the file for the conflict marker <<<<<<<. When you open the file in your text editor, you'll see the changes from the HEAD or base branch after the line <<<<<<< HEAD. Next, you'll see =======, which divides your changes from the changes in the other branch, followed by >>>>>>> BRANCH-NAME. In this example, one person wrote "open an issue" in the base or HEAD branch and another person wrote "ask your question in IRC" in the compare branch or branch-a.

8. If you have questions, please
   <<<<<<< HEAD
   open an issue
   =======
   ask your question in IRC.
   >>>>>>> branch-a

9. Decide if you want to keep only your branch's changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches. Delete the conflict markers <<<<<<<, =======, >>>>>>> and make the changes you want in the final merge. In this example, both changes are incorporated into the final merge:
10. If you have questions, please open an issue or ask in our IRC channel if it's more urgent.
11. Add or stage your changes.
12. git add .
13. Commit your changes with a comment.
14. git commit -m "Resolved merge conflict by incorporating both suggestions."

You can now merge the branches on the command line or push your changes to your remote repository on GitHub and merge your changes in a pull request.

Removed file merge conflicts

To resolve a merge conflict caused by competing changes to a file, where a person deletes a file in one branch and another person edits the same file, you must choose whether to delete or keep the removed file in a new commit.
For example, if you edited a file, such as *README.md*, and another person removed the same file in another branch in the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

1. Open Terminal.
2. Navigate into the local Git repository that has the merge conflict.
3. cd *REPOSITORY-NAME*
4. Generate a list of the files affected by the merge conflict. In this example, the file *README.md* has a merge conflict.
5. git status
   # On branch master
   # Your branch and 'origin/master' have diverged,
   # and have 1 and 2 different commits each, respectively.
   # (use "git pull" to merge the remote branch into yours)

```
# You have unmerged paths.
#  (fix conflicts and run "git commit")
#
# Unmerged paths:
#  (use "git add/rm ..." as appropriate to mark resolution)
#
#  deleted by us:   README.md
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

6. Open your favorite text editor, such as Atom, and navigate to the file that has merge conflicts.
7. Decide if you want keep the removed file. You may want to view the latest changes made to the removed file in your text editor.
8. To add the removed file back to your repository:
9. git add README.md
10. To remove this file from your repository:
11. git rm README.md
    README.md: needs merge
    rm 'README.md'

12. Commit your changes with a comment.
13. git commit -m "Resolved merge conflict by keeping README.md file."
    [branch-d 6f89e49] Merge branch 'branch-c' into branch-d

You can now merge the branches on the command line or push your changes to your remote repository on GitHub and merge your changes in a pull request.

Removing the Git-Ignore files from the Repository

One of the reasons might be that you already committed that files to your repository. In that case adding them to .gitignore doesn't prevent git from tracking these files. To fix that you should first remove them from repository like that:

git rm -r target/

(If your project has modules, you need to include the path down to the relevant module's target: git rm -r parent_project/module_name/target/)

If you want to remove directory from repository but not from disk you can use --cached option:

git rm -r --cached target/

In any case you should then commit that deletion to the repository (you can do it in the same commit with adding target/ to .gitignore if you wish).