GitHub & Git overview

Hyunjun, LEE

Department of Statistics

Korea University

Aug.19, 2019

Outline

GitHub as a personal storage (Naive approach)

Creating a new repository

- User can make new repository(folder) after creating new GitHub account
  by click on "Create a repository" or "New" buttons

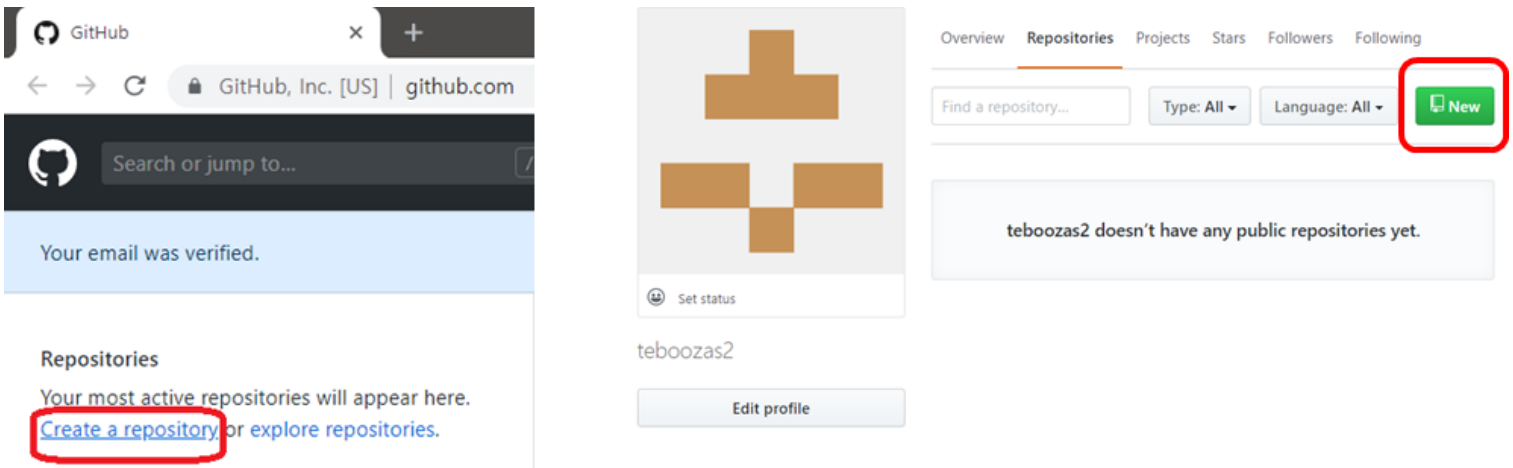


Figure: two ways to create new repository

Creating a new repository

- Name of a repository is url as well (need to be careful)
- Filling out README file and description details are recommended

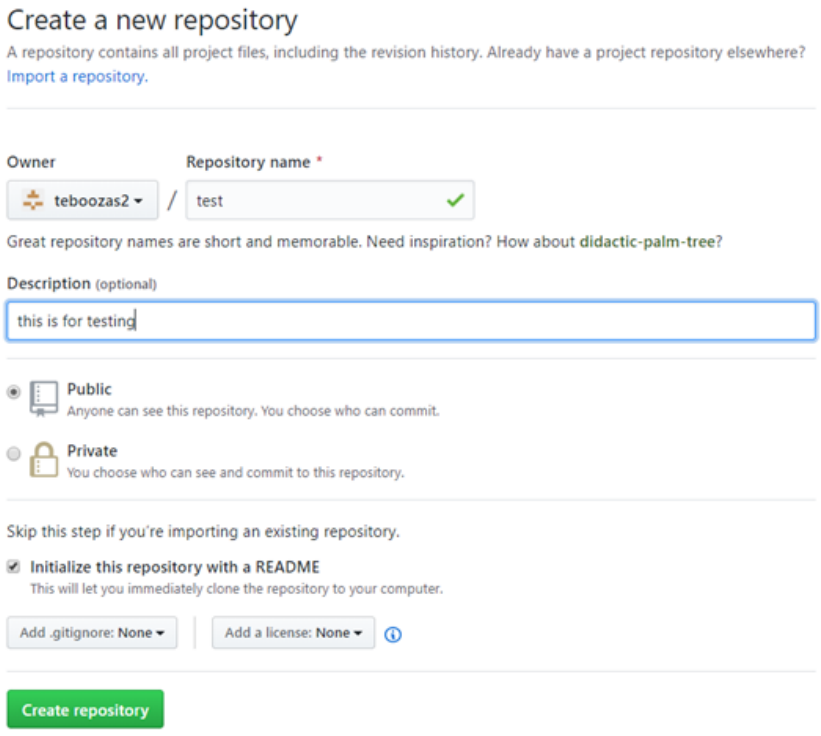


Figure: options for creating a new repository

Creating a new repository

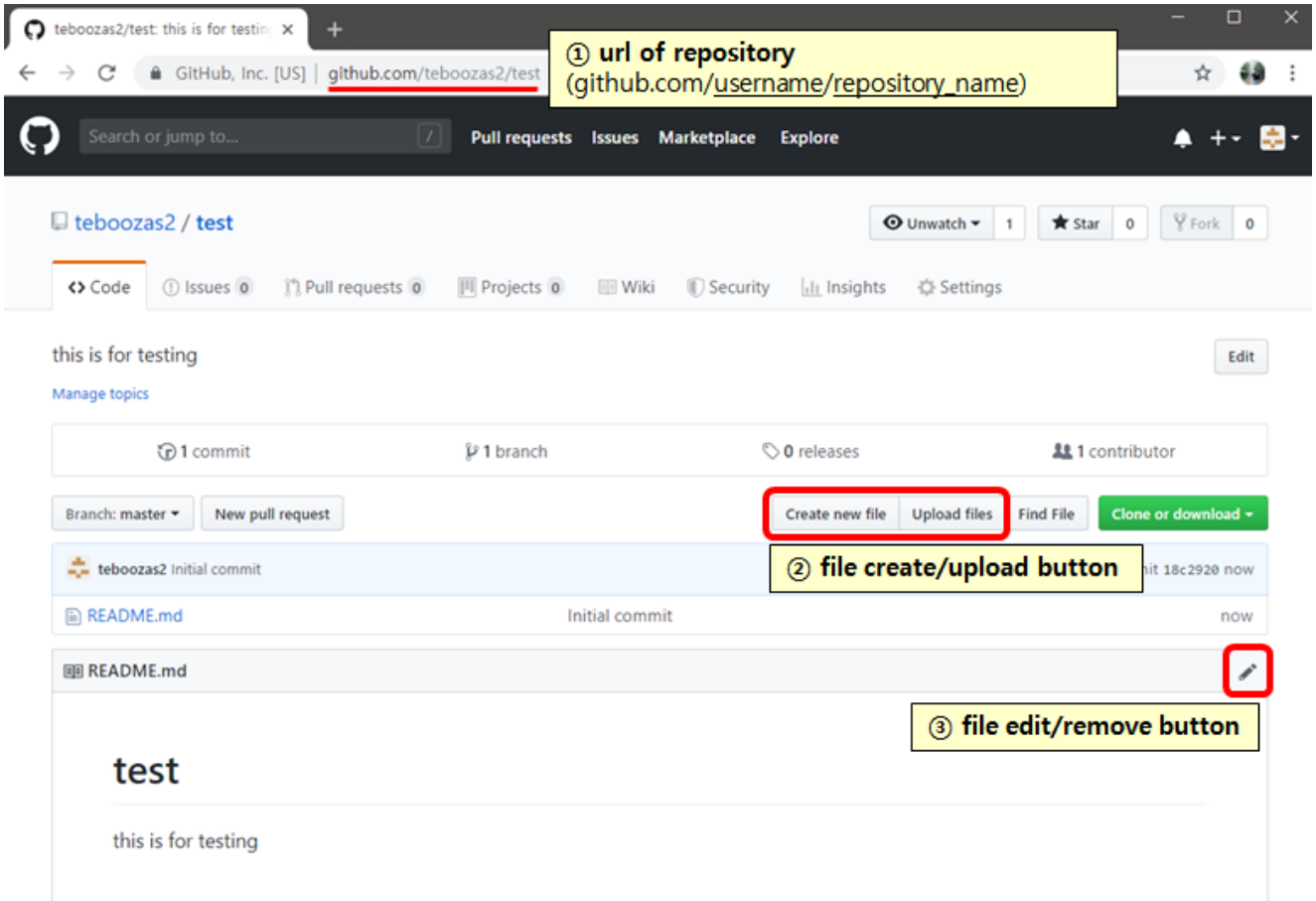


Figure: initial settings of a new repository

Upload, edit, and remove files

- To add files to repo, click "Upload files"
- Users can add items with drag up or to choose them manually
- Filling out commit message is strongly recommended
- After write down all messages, click "commit changes" to save files (we will check later what "commit" means)

Upload, edit, and remove files

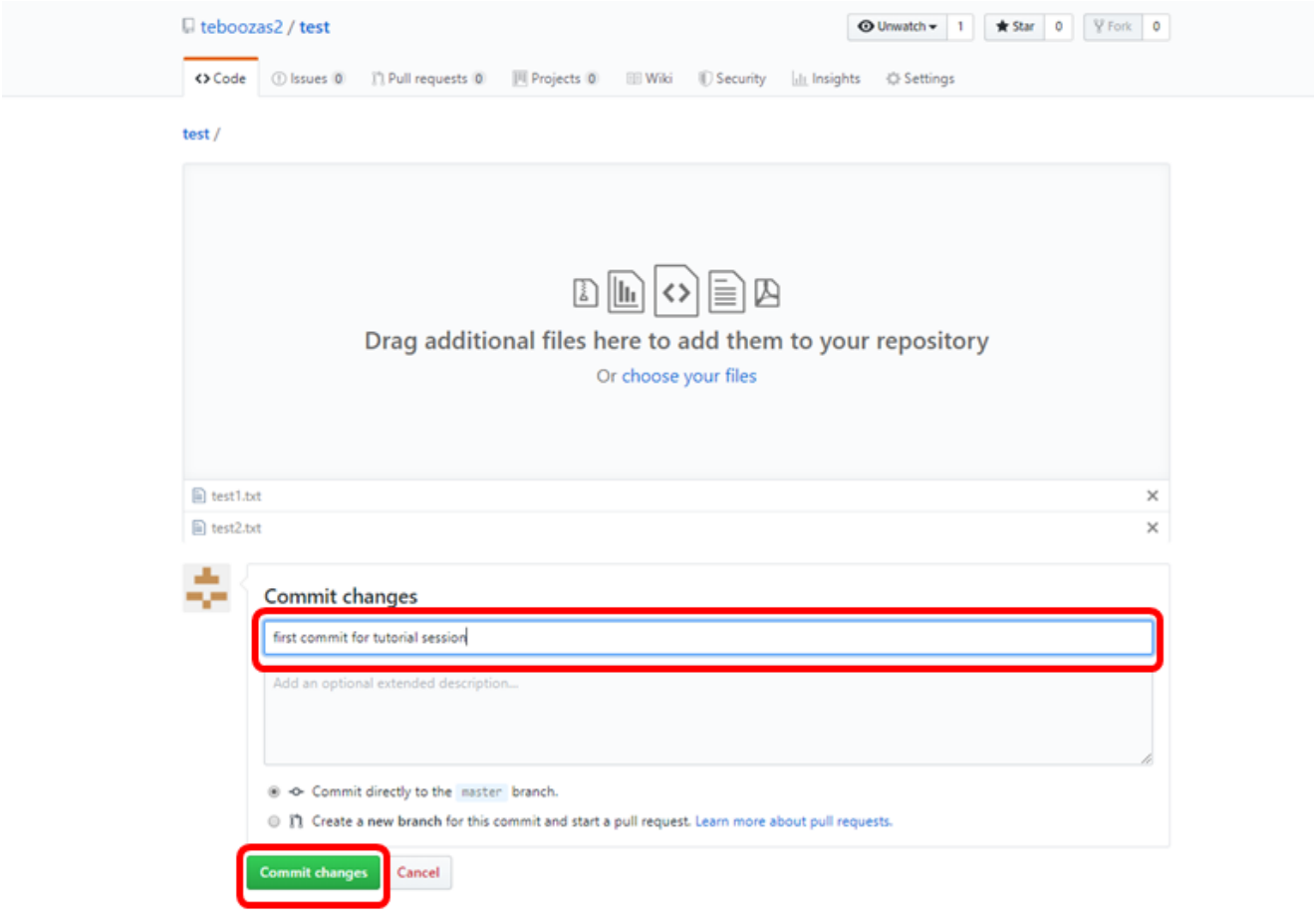


Figure: two text files are being uploaded

Upload, edit, and remove files

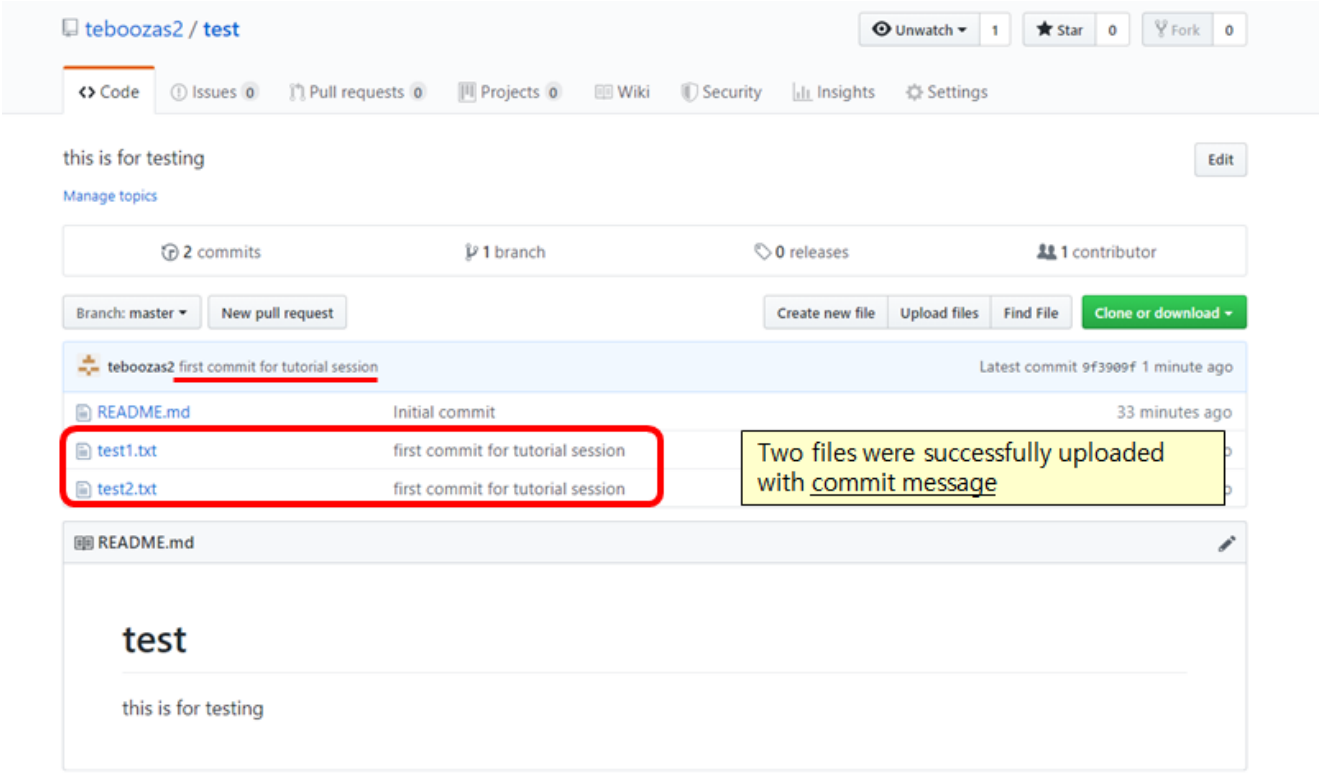


Figure: files were successfully uploaded with commit message

Upload, edit, and remove files

- Users can edit code or texts in repo directly on GitHub (not frequently used)

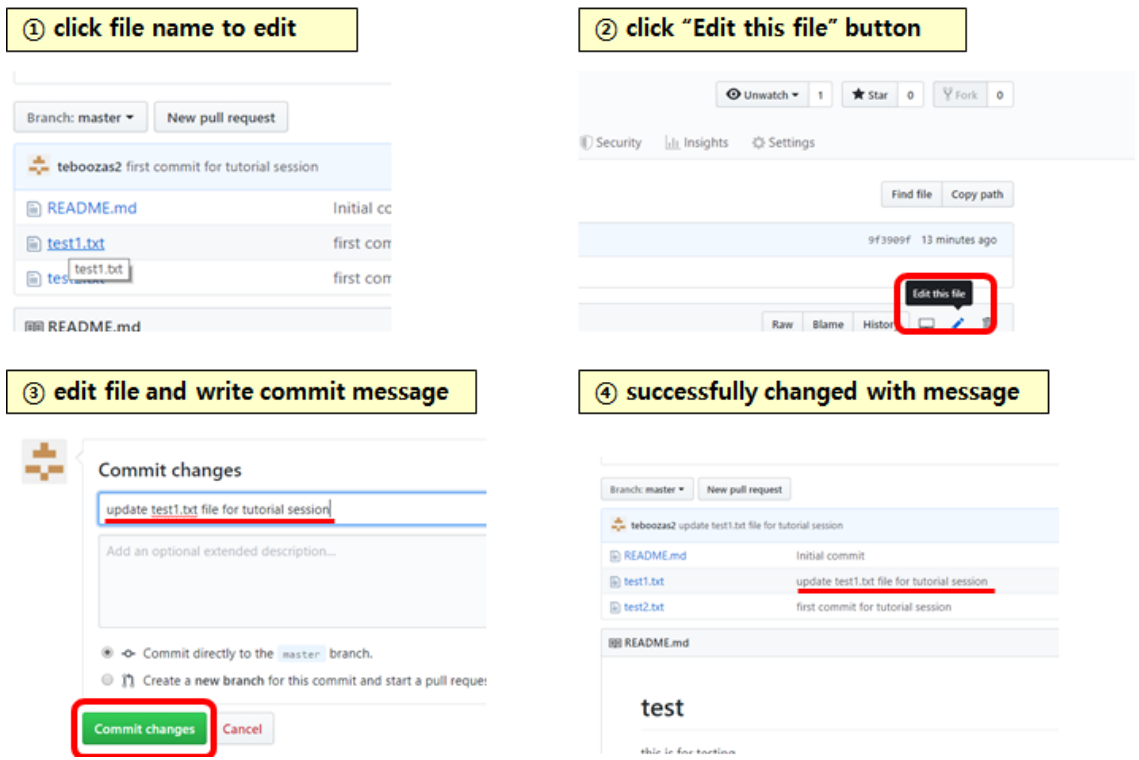


Figure: workflow editing files on GitHub

Upload, edit, and remove files

- Users can delete a file similarly with editing

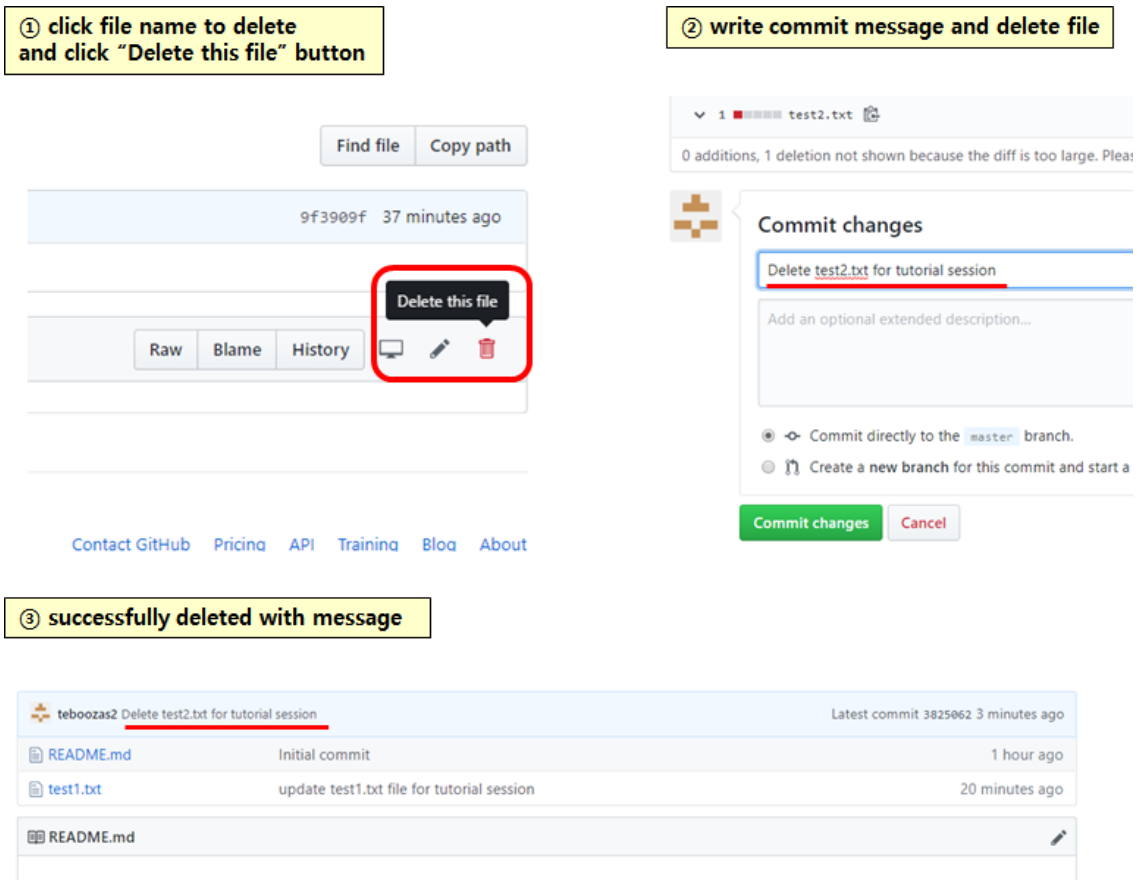


Figure: workflow deleting files on GitHub

More on GitHub repository

- rename, ownership transfer, removal of repository can be done in "settings"
- Changing histories of repository are all recorded and perfectly re-accessible
  - Files that edited and removed are preserved in each commit status
  - That's why commit messages are so important
    (to track records and access previous status of files if needed)
- Re-accessibility to previous status is powerful feature of Git (and GitHub itself)
- Although repo has no storage limit, maintaining it light is desirable (using local machine or cloud storage are better for large data)

More on GitHub repository

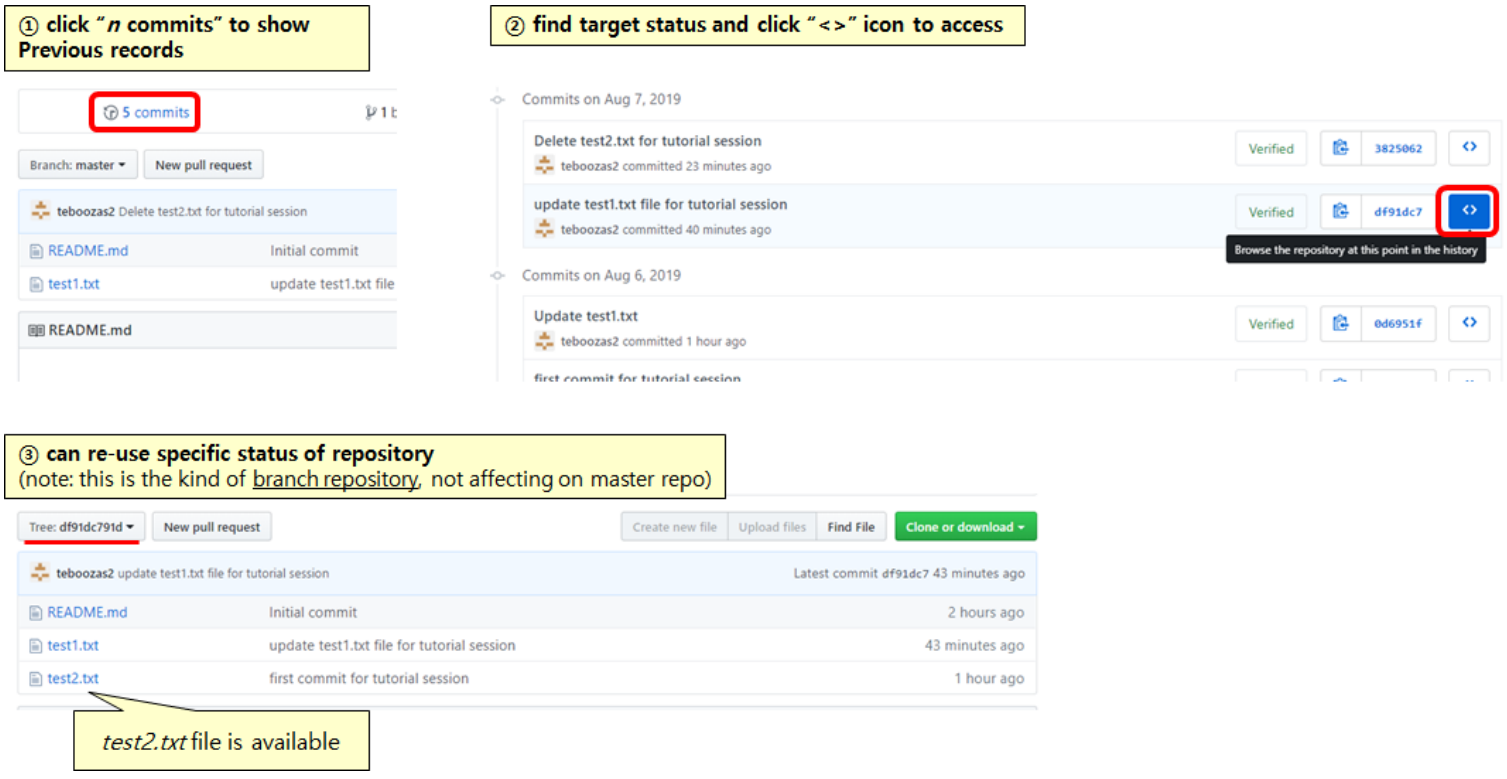


Figure: workflow accessing previous status of repo

More on GitHub repository

- What have been covered are enough for personal use and portfolio of own works
- However, GitHub is essentially the collaborative tool for developers based on Git software and Git repository
- For rich and interactive use of GitHub, understanding Git and its characteristic workflow is required

Quick Git - handling local repository

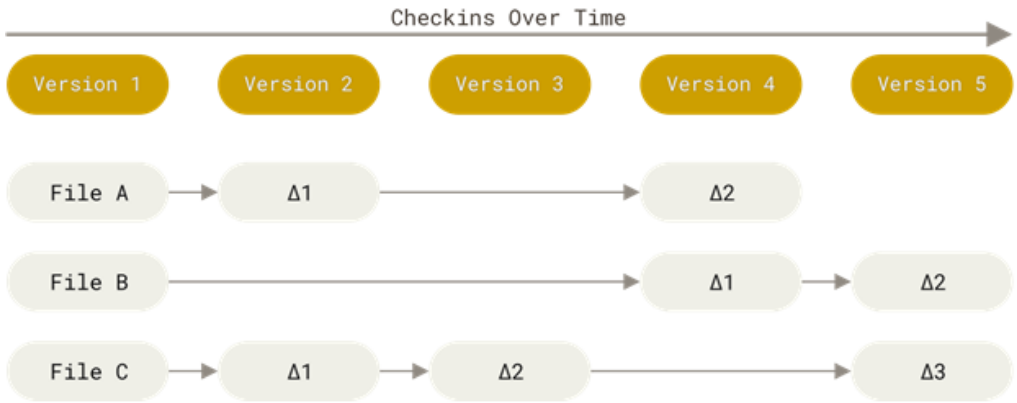
What is Git?

- Git is a version control system(VCS)
  - A version control system **records changes** to a file and files(folder)
  - Tracking records of files is useful for large scale collaboration
- Git is the most common VCS due to its speed, efficiency, and immensely large web hosting service, the GitHub

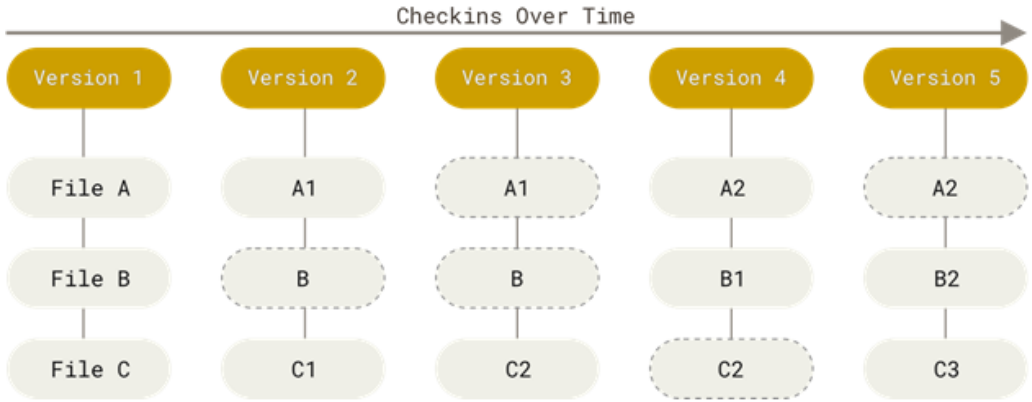
What is Git?

- Git takes 'snapshots' of status and records them
  - Most of VCSs track every single file change individually
  - In contrast, Git stores whole changes of folder into a small size of snapshot containing links of files
- Git has 3 main states of files (which is called 'tracked status')
  - **modified**: files in tracked status have been revised, not recorded
  - **staged**: modified files are on 'stage', ready to be recorded
  - **committed**: staged files are recorded into a snapshot **with checkpoint**
  - Note that not every files in Git repo are in committed state (untracked, modified, staged files don't have snapshots)

What is Git?



<delta-based VCS version stream>

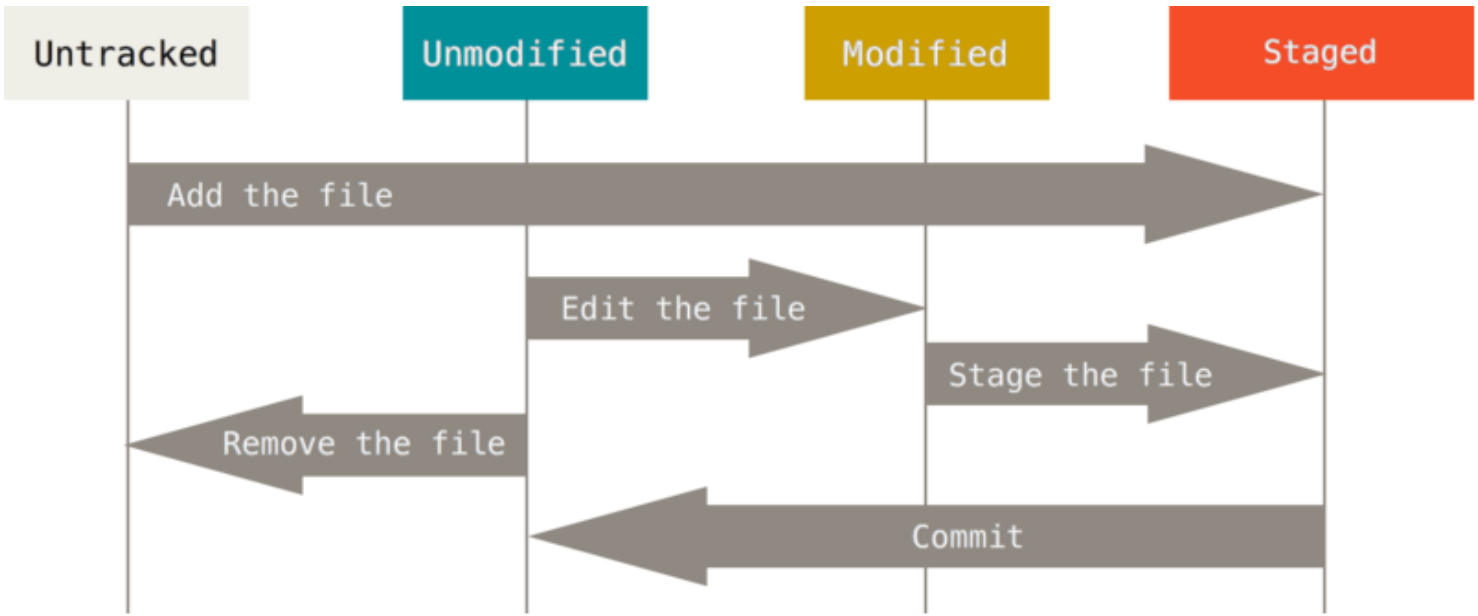


<snapshot-based VCS version stream (**Git**)>

Source: "ProGit", 2nd edition

Lifecycle of Git

- Summary of Git lifecycle(workflow) is depicted below
  - tracked status: unmodified(**committed**), modified, staged states
  - untracked status: files in local folder, but not be tracked by Git



Source: "ProGit", 2nd edition

Basic commands for Git CLI

- Most common tool for handling Git on local machine is CLI
  - CLI(Command-Line Interface) includes Git bash(Windows), Terminal(Mac), UNIX/LINUX shell prompt, etc.
- Users can use Git with CLI after installing Git on own machine (you can get Git from here: `https://git-scm.com/downloads`)

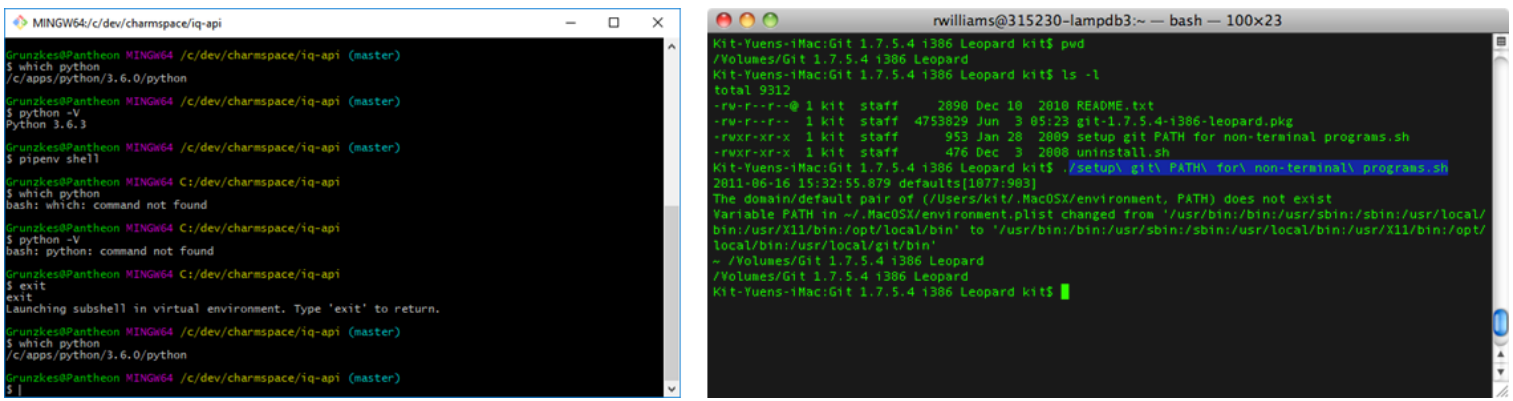


Figure: example of CLI

Basic commands for Git CLI

- After run CLI w.r.t types of machine, handling Git can be done with this basic form of command statement:

```
git <command> (options)
```

- Important commands can be categorized as below:
  - configuration & settings - `config` , `help`
  - making a Git repository - `init` , `clone`
  - snapshot handling - `add` , `status` , `commit`
  - branching - `branch` , `checkout` , `merge`
  - sharing projects - `remote` , `fetch` , `pull` , `push`
- first three categories will be covered in this section (and remainders for later sections)

Basic commands for Git CLI - **configuration & settings**

- `config` command (examples)
  - command for configuration, including user information, etc.
  - `git config --global user.name "John"` :
    set user name as 'John'
  - `git config --global user.email john@example.com` :
    set user e-mail as 'john@example.com'
  - `git config --list` : check global settings of local Git
- `help` command (examples)
  - roughly two ways to get help for sepcific command
  - `git help config` :
    get the manpage help for the `config` command
  - `git add -h` :
    git summary help for the `add` command directly from the kernel

Basic commands for Git CLI - **making a Git repository**

- `init` command
  - turn current directory(folder) which CLI is running on, into a Git repository (can store snapshots)
  - `git init` (just simply enter it!)
- `clone` command
  - get a clone(copy) of existing Git repository, including all of the previous snapshots
  - `git clone https://github.com/John/myrepo Johnrepo` : copy Git repo 'myrepo' from url, and change its name into 'Johnrepo'
  - `git clone` is useful statement, especially for Google Colab

Basic commands for Git CLI - **snapshot handling**

- `add` command (examples)
  - turn modified & untracked files into staged files
  - `git add README.txt` :
    send 'README.txt' file(untracked or modified) to staged state
  - `git add -A` :
    send all of the untracked/modified files in Git repo to staged state
- `status` command (examples)
  - get a view of states of all files in Git repo (including untracked ones)
  - `git status` : return list of files by states in the repo
  - `git status -s` : return short version of status

Basic commands for Git CLI - **snapshot handling**

- `commit` command (examples)
  - take a snapshot of status of Git repo and store it with a checkpoint
  - `git commit -m "initial commit"`:
    take a snapshot of status with commit message 'initial commit'
  - `git commit -a -m 'second commit'`:
    take a snapshot of status with commit message 'second commit', skipping staging step (untracked/modified files are automatically committed without `add` command)
- Notes on `commit`
  - After running `commit` command, all committed files are turned into unmodified state (recall 'the lifecycle of Git')
  - Every commits have their own checkpoint, called 'SHA-1 checksum' (looks like `3c163a0`)
  - Users can revert to or compare snapshots with checkpoints

Notes on basic commands for Git CLI

- If you were confused, recall the depiction of 'the lifecycle of Git'
- Although there are lots of commands and options, `add` and `commit` are the fundamental ones
- Past checkpoints of commit contains all files at the time, even they were deleted and do not exist now on (very useful for recovery)
- The reason why Git separates states into 'staged' and 'unstaged' is that, users can record changes 'as the way of project' (rather than save every single changes of each files)

Quick Git - branching

What is branching?

- Branching method allows running independent subprojects
  - Developers can work independently(debugging, add functionality, etc.) without concern of conflicts exploiting branches
  - Branching is the core of DVCS(Distributed VCS) for massive collaborative projects
- Branching of Git is easy and fast enough to be beloved

Figure: example of branching workflow

What is branching?

- Generally, branching follows workflow below:
  1. create branch repository with purpose from master branch
  2. move onto created branch, and do add/commit to achieve purpose
  3. after commit, move back to master branch and merge outputs of target
- Each step above is matched with the corresponding commands:
  `branch`, `checkout`, `merge`

Commands for Git branching

- `branch` command (examples)
  - create & list branches from the other branch (mostly 'master')
  - `git branch debug` : create 'debug' branch based on current branch
  - `git branch` : return existing branches based on current branch
- `checkout` command
  - switch branches for working on
  - `git debug` : moves to 'debug' branch for working on
- `merge` command
  - merge outputs of a branch with the other's
  - `git merge debug` :
    (automatically) combine files in current branch with files in 'debug' branch (current $\leftarrow$ 'debug')

Notes on commands for Git branching

- Even though commands and their workflow seems quite simple, Git branching can be extended for numerous non-linear sub-projects
- In case of conflict(fail to merge), users can fix conflict manually and commit after fixing, or using `mergetool` command
- Users can review outputs in branch before merge them to master
  $\Rightarrow$ this forms basis of 'pull requests' in GitHub service

Quick Git - sharing projects (remote repository)

Remote repository and web hosting service(GitHub)

- Most of the projects using VCS(including Git) store files and commit checksums in remote repository
  - remote repository is usually hosted on the Internet or network
- There are many web hosting services providing remote repository and the most famous one is **GitHub**
- Communication between local Git and web-based GitHub can be easily done with local CLI and its commands

Remote repository and web hosting service(GitHub)

- workflow to control remote repo can be summarized:
  1. add(link) or check remote repository on local machine
  2. get all data from origin remote and merge them with local data
  3. after fixing merged local data, merge them toward origin remote
- Each step above is matched with the corresponding commands: `remote`, `fetch` or `pull`, `push`

Commands for controlling remote

- `remote` command (examples)
  - link & list remote(mostly called 'origin') from hosting service
  - `git remote add origin https://github.com/John/myrepo` : add(link) remote 'origin' from given url of hosting service
  - `git remote` : return existing linked remote repositories
- `fetch` command (examples)
  - get data from remote (except for things already exist)
  - `git fetch origin` : get data from 'origin' remote to current repo

Commands for controlling remote

- `pull` command (examples)
  - do 'fetch' and 'merge' at the same time
  - `git pull origin master` :
    get data from 'origin' remote to current repo('master') and merge them within 'master'
- `push` command (examples)
  - share local commits with the origin remote repository
  - easily can be thought as saving data from local to GitHub repository
  - `git push origin master` :
    share commits in 'master' repo with 'origin' remote repo

Notes on commands for controlling remote

- `pull` and `push` are key commands for open-source projects
  - These two commands allow interactive workflow between local and remote repository
  - Combining with branching, `pull` and `push` commands become more powerful
- `clone` versus `remote add`
  - `clone` command get entire repository into local machine, but this copied repo is not linked with origin remote
  - On the other hand, `remote add` command just link local machine with web-based repository, and not pull down commits and data

GitHub as a collaborative tool for open-source projects

Review: snapshotting, branching, and remote control

- Combining all, working on a Git project can be summarized:
  1. create a project on GitHub repository
  2. make a Git repository on local machine for maintenance of the project
     - `init`, `clone`
  3. connect local repository with remote GitHub project repository
     - `remote`
  4. get previous commits from remote repository
     - `fetch` or `pull`
  5. develop(snapshot) a project on local with related commits from remote
     - `add`, `status`, `commit`
  6. share additional commits from local to Github repository
     - `push`
  7. if needed, use branching for independent works
     - `branch`, `checkout`, `merge`

Working on a open-source projects

- As mentioned before, GitHub is widely applied on large number of open-source projects
- Thus, GitHub provides functionality for teamwork onto projects owned by group of users, as well as for potential contributors
- For this, GitHub came up with the concept of **Pull Request**

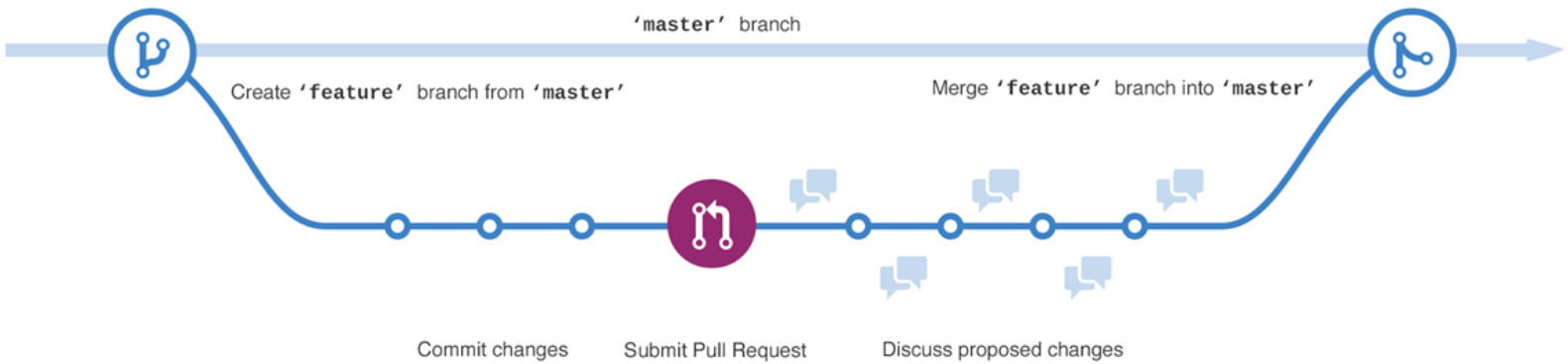


Figure: standard GitHub workflow

Working on a open-source projects

- If someone wants to contribute to a open-source project via GitHub, followings are the standard steps
  1. **Fork** target GitHub project into personal account
  2. clone and link(remote) it to own local machine using CLI
  3. create a new branch to work with
  4. add commits to contribute to the project (still on a branch)
  5. push commits in branch toward origin remote (not to master branch)
  6. click **Compare & pull request** button to submit pull request
  7. project owners will review and discuss upon pull requests, and decide whether to merge this pull request or not
  8. if accepted, project owners will merge commits in the pull request
  9. pull updated project into local repo and delete previous branch
  10. repeat 3. to 9. for additional contribution

Working on a open-source projects

- Shortly, **pull request** is the expression of asking to merge user's commits with codes in the open-source project
  (and darely described as 'the heart of collaboration on GitHub')
- You can be a contributor for named GitHub project with this process, including NumPy, Scikit-learn, Tensorflow, PyTorch, etc.

Conclusion

- You don't have to get to know all of these at once.
  Instead, try to get used to GitHub and Git CLI (using toy examples)
- At first, it is desirable to make a habit of managing GitHub account "regularly", even with the purpose of personal storage
- Keep track of concepts of version control and collaborative open-source project. It will surely be helpful to both academic achievement and successful career.

Reference

- ProGit book, 2nd edition (Chacon, Straub) (link)
- GitHub Guides official webpages (link)
- Opentutorials.org - Git from hell (link)
- Korean blogger(developer) - Pull Requests (link)