GitHub Guide for the Structures and Multidisciplinary Optimization Group

Brian Burke and Yicong Fu

May 2022

1 Overview

GitHub is an online service that hosts tools for software development and version control using Git. Git is an open-source software that is commonly used for coordination among programmers during software development. Git offers distributed version control and source code management to enable programmers to work on the same project across various branches of code and on various machines. Git can be installed from `https://git-scm.com/downloads`. Git is commonly used as a command line tool; a cheat sheet from `https://education.github.com/git-cheat-sheet-education.pdf` has been included as an appendix to this document. Note that GitHub graphical user interfaces are also available as downloads for Windows and Mac–see the same cheat sheet for respective links.

1.1 Git

There are two main ingredients of git: the remote repository and the local copy. The remote repository is the server where a code is stored. It is where you can get the code from by *pulling* or upload your changes by *pushing*. The local copy is the local repository stored on your workstation. You can make changes to this local copy using a code editor or terminal interface. Changes are then added to the history of your local copy through *commits*, which help to organize changes to the code in a log format. The remote repository is only updated when you *push* to it. Once you feel comfortable with the changes you've made locally, you can *push* your *commits* to the remote repository to share your progress with your collaborators. Other people can then *pull* your changes to their own local copies to keep their code updated.

Basic git only has command line interface (CLI). If you need a GUI you could download GitHub Desktop. In terms of how git works and how to use git CLI, there are many good references out there. The following ones are a few beginner-level tutorials:

- https://missing.csail.mit.edu/2020/version-control/
- https://learngitbranching.js.org/

1.2 GitHub

Git repositories can be hosted either on local machine, on some server in the local network, or on the internet. GitHub is one of the biggest service providers of internet hosting for git repositories. There are two main versions of GitHub: public (github.com) and enterprise (e.g., github.gatech.edu). These are separate from each other. However, the SMDO group will generally be using the public GitHub only. The following sections will cover the new workflow.

2 Workflow

2.1 Prerequisites

First, make sure you have an account for public GitHub and git on your local machine, which is usually preinstalled on Unix-like systems (but not for Windows).

Next, register this account on your local machine to let git know who you are. This can be done by running the following commands in terminal from anywhere:

```
git config --global user.name your-account-name
git config --global user.email your-account-email
```

The commands above will write account information to your global git configuration file, which is in `$HOME/.gitconfig` for Linux and MacOS. Notice that if you also have other projects under a different github account (e.g., another github.com account or an enterprise account), you will need to set it up separately by running the following commands in the project folder in terminal:

```
git config user.name your-account-name
git config user.email your-account-email
```

These commands essentially write to a repository-specific configuration file located in:

`project-folder/.git/config`.

In order to prevent some issues when pulling, you will also need to do run:

```
git config --global pull.ff only
```

2.2 Fork and Pull Method

The SMDO group uses the *Fork and Pull* model for collaborative development. Under this method, anyone can fork an existing repository to their own account where they are free to make changes to their personal fork. The repositories on the `https://github.com/smdogroup` organization page serve as the base/parent repositories for all major software projects under development in the SMDO group. Individuals may make changes and commits as necessary to get new code working. Once your personal fork is working and passes any applicable tests, you can open a *pull* request from your fork to the parent repository. The pull requests allows users to discuss and review changes to the code before merging into the base repository. After a successful review by one of the project maintainers, the pull request will be merged into the main branch.

1. You should work from your own fork of the main `github.com/smdogroup` repositories. This is easily done using the web interface and clicking the "fork" button on the respective repository. This will create a separate repository under your own account.

2. Clone your fork to your local machine. Check for any references to other repositories with `git remote -v`; delete these extraneous references. Set the origin as your personal GitHub fork. Notice that "fork" is a GitHub specific feature which git doesn't have. So once you clone the repo, anything you do with `git` command will only affect your "forked" repo, but not the "upstream" repo under smdogroup's account. This can be verified by running:

   ```
   git remote --verbose
   ```

   And you will only see the urls of your own remote repo (but not the repo under smdogroup).

3. Create a new branch under your repo for the feature you'll be working on. The name of the branch should be precise and self-explanatory. This can be done either on the GitHub webpage or locally by running the following command **on the branch you're branching from (for example master)**:

```
git checkout -b name-of-your-branch
```

which will create a new branch and switch to it. Then you can do:

```
git branch
```

to verify that you're at the correct branch.

Note: Branching is implemented in a very cheap way for git, and it makes the code base organized if many different features are developed at the same time. so it's a good practice to always make a branch for certain task. It is discouraged to commit directly to master (or main, which is GitHub's new nomenclature) branch, even for your own forked repository.

4. Now you are ready to work on the code. You can commit as many times as you want to your branch. **Before making changes**, be sure to pull from the main github repo to your fork. Then pull to your local copy using `git pull origin` or `git fetch origin`. This is done in case there are new changes in the remote repository or the main repository that you don't have locally.

   It's recommended to frequently push local changes to GitHub, so that you will always have backups online. You can only push *commits* (but not uncommitted changes) so make sure to commit before push.

5. While we are working on the local fork, the upstream repo might change. Therefore, you need to periodically update your local fork. This needs to be done on the GitHub webpage, by simply clicking the "Fetch upstream" button. If there are indeed updates from upstream, you might also need to update your branch. This is because your branch was originated from some other branch (likely the master), which might be updated by such action. You can update your branch by:

   ```
   git checkout name-of-your-branch && git merge branch-to-update-from
   ```

6. Once you feel comfortable with your changes, it's time for the main repo to incorporate your contributions! This will be done via "pull request", which is another GitHub specific feature that git doesn't have. So you will need to go to the GitHub webpage and open a pull request ticket. Although the term might be confusing, it can be thought of as "to request a pull to main from you". The pull request should be from your repo to smdogroup/repo, and usually it'll be to the master branch. The pull request then will be reviewed, tested, and finally approved and merged (hopefully).

   To make it easier for the project maintainers (those who will be reviewing your pull requests), it is important to **include thorough documentation in your pull requests**. Inside a pull request, it is also possible to make new commits, write comments, and add replies. In this way, a pull request can form a place to discuss proposed changes before they are implemented in the main repository.

7. Once the pull request is approved, you should delete the local branch you've been working on. Never keep working on the merged branch and "pull request" again in the future, because it and will cause redundant commit history to the master branch.

8. Lastly, you will do a final "Fetch upstream", then your work will be reflected in the master branch of your own forked repo. And do another `git pull` locally to download the updates.

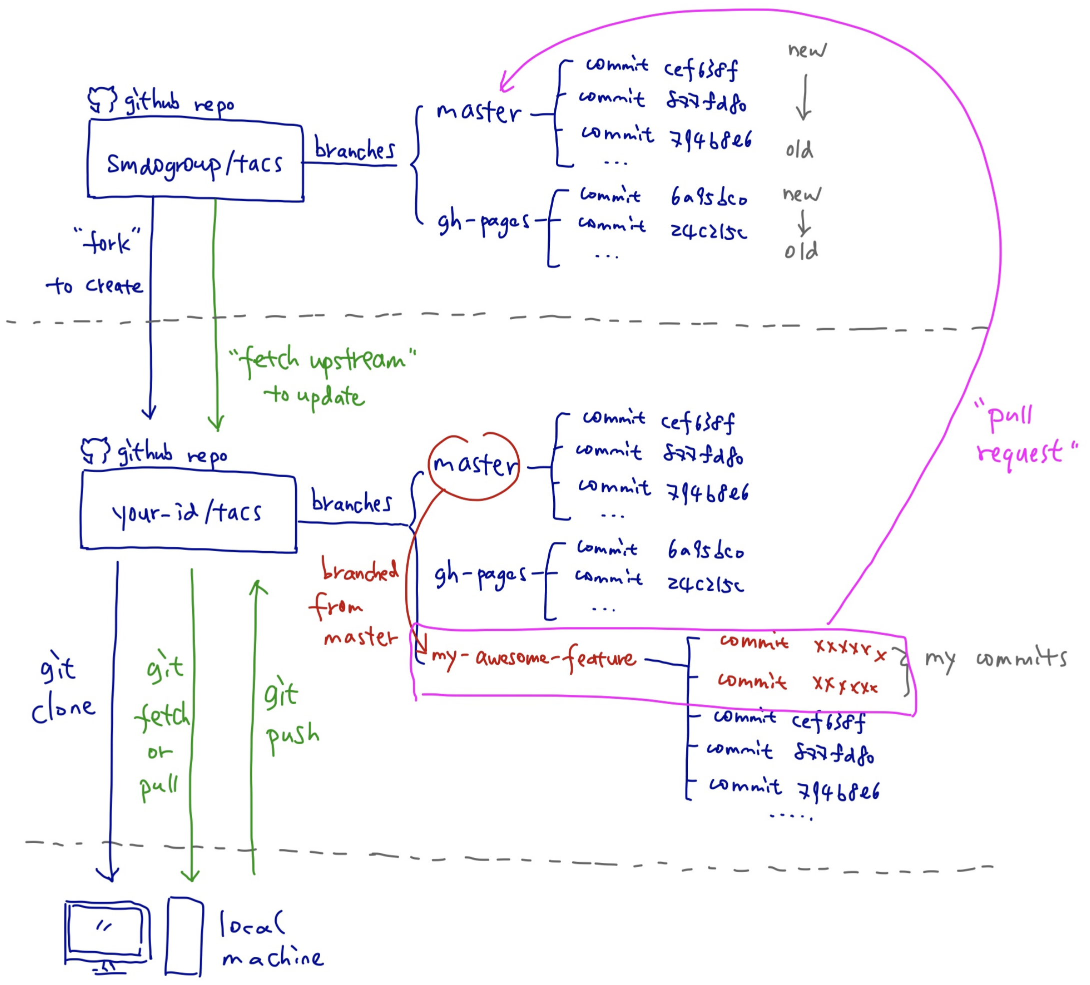

Figure 1: Git/github workflow diagram

GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

```
git config --global user.name “[firstname lastname]”
```

set a name that is identifiable for credit when review version history

```
git config --global user.email “[valid-email]”
```

set an email address that will be associated with each history marker

```
git config --global color.ui auto
```

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

```
git init
```

initialize an existing directory as a Git repository

```
git clone [url]
```

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

```
git status
```

show modified files in working directory, staged for your next commit

```
git add [file]
```

add a file as it looks now to your next commit (stage)

```
git reset [file]
```

unstage a file while retaining the changes in working directory

```
git diff
```

diff of what is changed but not staged

```
git diff --staged
```

diff of what is staged but not yet committed

```
git commit -m “[descriptive message]”
```

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

```
git branch
```

list your branches. a * will appear next to the currently active branch

```
git branch [branch-name]
```

create a new branch at the current commit

```
git checkout
```

switch to another branch and check it out into your working directory

```
git merge [branch]
```

merge the specified branch’s history into the current one

```
git log
```

show all commits in the current branch’s history

INSPECT & COMPARE

Examining logs, diffs and object information

```
git log
```

show the commit history for the currently active branch

```
git log branchB..branchA
```

show the commits on branchA that are not on branchB

```
git log --follow [file]
```

show the commits that changed file, even across renames

```
git diff branchB...branchA
```

show the diff of what is in branchA that is not in branchB

```
git show [SHA]
```

show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

```
git rm [file]
```

delete the file from project and stage the removal for commit

```
git mv [existing-path] [new-path]
```

change an existing file path and stage the move

```
git log --stat -M
```

show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

```
git config --global core.excludesfile [file]
```

system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

```
git remote add [alias] [url]
```

add a git URL as an alias

```
git fetch [alias]
```

fetch down all the branches from that Git remote

```
git merge [alias]/[branch]
```

merge a remote branch into your current branch to bring it up to date

```
git push [alias] [branch]
```

Transmit local branch commits to the remote repository branch

```
git pull
```

fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

```
git rebase [branch]
```

apply any commits of current branch ahead of specified one

```
git reset --hard [commit]
```

clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

```
git stash
```

Save modified and staged changes

```
git stash list
```

list stack-order of stashed file changes

```
git stash pop
```

write working from top of stash stack

```
git stash drop
```

discard the changes from top of stash stack