NEIC Gitlab Developer Workflow

Table of Contents

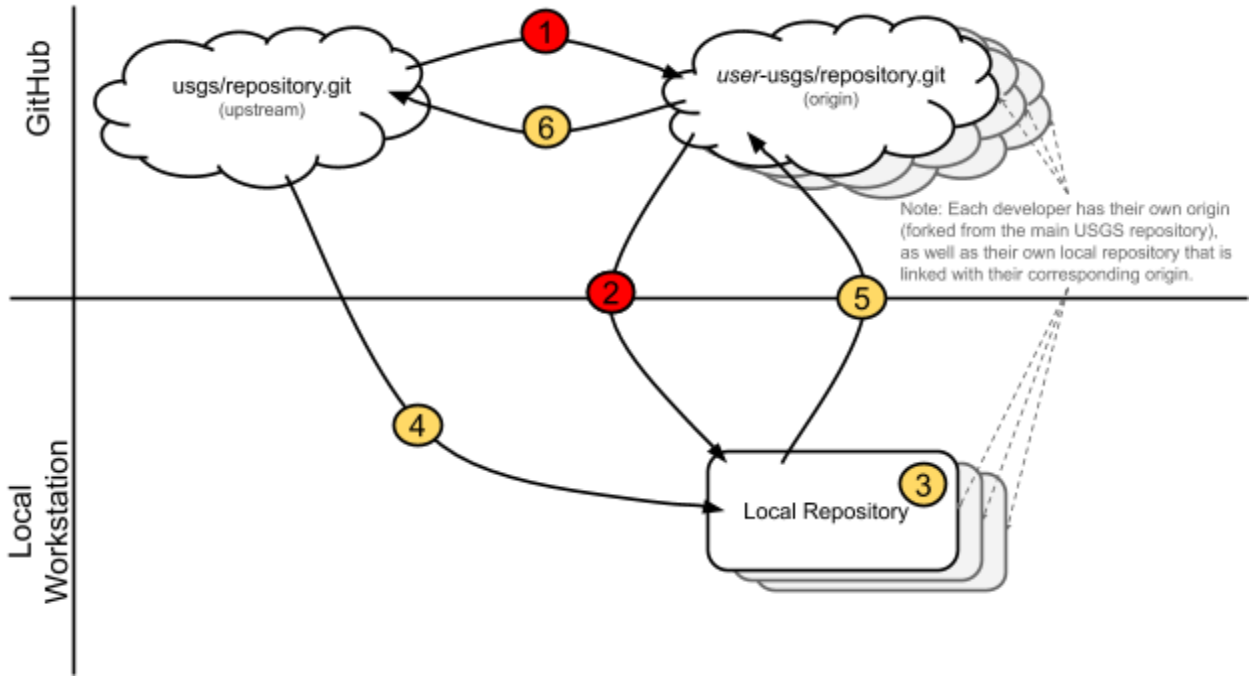
Development workflow

If you do not yet have a gitlab account, see Set up your gitlab account. The steps immediately below describe how to start contributing to an existing gitlab project. If you are creating a new project repository, see Creating a new git repository. If you would like to get more confused see these background References.

Overview

The following bullets outline the process and will make more sense once you read the rest of the document and experiment with git.

- Developer codes and commits locally to a feature branch
- Developer pulls changes from the upstream into the local feature branch
- Developer pushes feature branch back to origin
- Developer creates pull request from origin/feature → upstream/master
- Note, developer only pushes to origin and pulls from upstream. Updates to the upstream master are done through merge requests. This maintains the integrity of the upstream master which is always in a deployable state.
- Coding is never done on the local master branch. Always code on a local branch you create.



Detailed Workflow: (Yellow and red numbers match steps in below.)

Create repository links and getting code

Steps one and two only have to be done once for each project

- **ONE** Developer forks project's "upstream" repository to user's "origin" repository
    - This is done via the git website with fork button - github

- **TWO** Developer clones user-usgs/repository.git to local files system
    - git clone <url> (url of your forked copy)% This creates folder with repo name with master branch and origin remote that refers back to the clone url
    - Establish connection (alias) to upstream repository
        - cd to the project directory (that was just created from clone call)
        - git remote -v % List the current configured remote repo for fork
        - git remote add upstream <url> (url of the original)% Link to new remote upstream repo to be synced with users repositories. Url is obtained from upstream project on git website
        - git remote -v % Verify that new upstream repo has been established

Changing, writing, and saving code to your local and origin repos

Steps three to six are done throughout development process.

- **THREE** Developer creates a new local branch for making changes to code. The step is repeated until a new deployable version of the code has been developed. Never make changes to local master.
    - git checkout -b <branch_name> % <branch_name> is working, BugFix, etc.
    - git branch % verify working branch

- Developer commits changes made to branch on local file system
    - git add <file> % Can use “git add .” rather than list individually
    - git commit -m ‘____’ % Creates message describing changes made
    - The above two command can be combined with: git commit -a -m ‘comment’
    - git push origin <branch> % Save changes to <branch> to gitlab origin

Saving and sharing code updates to upstream repo

- **FOUR** Update your local branch from upstream master. This will incorporate any changes to master that were done by others.
    - git status to make sure all changes were committed to local feature-branch.
    - git checkout master % Switches you to point at the master branch (note: checkout can also be used to revert files to their original state of last update)
    - git pull --ff-only upstream master % fast forward, will fail if conflicts are found between local and upstream master, I think. (John uses git pull --rebase upstream master see below)
    - git push origin master % syncs origin master to upstream master. Really don’t need this unless working in a complex environment. We never use origin master for anything.
    - git checkout <branch> % Switches you to point at your local branch (the one with the changes you want to add to the)
    - git rebase master % this will incorporate any changes to upstream master to your local branch.
        - If there are conflicts - follow steps below till all fixed. Might want to talk to John if this happens to walk you through it.
            - git status
            - edit file in text editor of choice (<<<HEAD === >>>MSG)
            - git add <file>
        - git rebase --continue % Continues rebase after conflicts resolved?
        - git rebase --skip % Will skip over file with conflicts
        - git rebase --abort % Will restore all files to pre-rebase

- **FIVE** Developer pushes branch changes to origin
    - git push origin <branch> % Creates a feature branch in origin with commits <branch> is the name of the branch with the wanted changes in it. This is NOT master. If local branch is too out of sync you can force an update: git push --force origin <branch>
    - check origin for new branch on git website

- **SIX** Developer issues pull request to merge commits from origin feature-branch to upstream master branch
    - On the git website, switch to feature-branch then make pull request by hitting the “Create merge request” button. On gitlab click “merge” in left panel (don’t issue merge request for master branch)
    - You can select a button to have gitlab to delete origin feature-branch immediately after it has been accepted by the authority of upstream master branch.

- Authority of upstream master branch (generally someone besides you) will then merge pull request via git website. To make sure the Authority sees your request you can assign the task to her or him via the gitlab website.

- Resync local and upstream master after pull request has been accepted.
    - git checkout master
    - git pull --ff-only upstream master
    - git push origin master

- Once changes have been accepted via the pull request the local and origin branches can be deleted.
    - git branch -D <branch> %Delete branch locally (STILL NECESSARY even with delete above that deleted origin branch)
    - Delete origin/repo branch on git website, branches page (THIS is the delete that is optionally done through the gitlab website when issuing the pull request, described above)

- **Make a new local branch so you don’t mess up master!**
    - **git checkout -b <branch> % ex. BugFix, working, etc**

Lingo

online glossary

- **repository:** A stand alone copy of the code, supporting files, history of changes, and other technical stuff.
    - **remote:** This is the version of a repository that is hosted on an external git server. It can be connected to local repositories so that changes can be synced.
        - upstream, deployable repository
        - origin, user’s remote repository
    - **clone:** A clone is a copy of a repository that lives on your computer.

    - **fork:** A fork is a personal copy of a repository that lives on your git remote account. A fork is a request for GitHub/GitLab to clone the project and register it under your username.
    - **branch:** A parallel version of a repository. It allows the user to diverge from the main line of development *within a repository* and continue to do work without messing with that main line (master).
        - **master:** Default branch. All repositories have a master branch. In this workflow, the upstream master is the deployable code and coding should never be done on the master branch in any repository, In order to maintain the master's integrity.
- **commit:**
    - A commit, or "revision", is an individual change to a file (or set of files).

Set up your gitlab account

In order to use git, you have to set up ssh keys so you can transfer files to and from gitlab. This only has to be done once for the lifetime of your account.

- Login to gitlab: You should have access to gitlab using your AD user/password here:
    - https://gitlab.cr.usgs.gov/users/sign_in
- Setup SSH key
    - If you are not familiar with .ssh keys ask someone to help you generate them on your local machine. Gitlab requires an RSA key e.g., ~/.ssh/id_rsa.pub
    - Cut and paste ssh key onto gitlab website user-icon->settings->ssh keys

Creating a new git repository

There are multiple ways to create a gitlab repository. You can initially create an empty repository using the gitlab website, clone that to your local computer then push your code to the website. Or you can create a git repository on your local computer and push it to the website. The latter is described below and neither way is preferred.

- If code exists but not in git repository
    - cd to existing code directory
    - **git init**
    - **git add .**
    - **git commit -m "First commit of existing code"** commit code to local repo
    - create new project on gitlab website and copy project's URL
    - **git remote add upstream <URL>** link to new project:
    - **git remote -v** verify link
    - **git push -u upstream master**
    - You will now need to follow steps one and two below to fork the upstream to a remote origin on the git website and link to that to your local code repository to follow workflow.

References

- Git Conceptual
- John Patton wrote a Git document that is more a conceptual than this document but also confusing.
- We follow the "integration manager workflow (second section) "
- Git glossary

Appendix

I think this describes rebase: