Brief Introduction to Github/Git Workflow

Create a Github Account and Token (one time setup)

In order to create repositories, you will need to have a Github account, and you will need a Personal Access Token to use for authenticating your account from the command line.

- Navigate to Github.com
- If you have an existing account, log in (MFA recommended)
- Choose to Create an account - use an email address you can remember, as you often use it on Github to find users. Once you create an account, log in.
- Under your profile image (upper right corner), select Settings
- Navigate to Developer Settings (bottom left) and then choose “Personal Access Tokens”
- Create a token, and save it to a secure location - this will act as your password!

Create a Fork of the repository (one time setup)

A “fork” of a repository is your own personal copy - it contains all the code / documents / assets of the original, including the complete history of changes, but you have the ability to modify it. Your fork does not affect the original repository.

- Ensure you are logged in to Github (check for your profile picture in the upper right corner)
- Navigate to the original repository URL provided
- Select the “Fork” button (tree icon) in the upper right
- Choose an Owner to fork into if necessary
- Choose a repository name (the default is fine)
- Select Create Fork (this may take a few seconds)
- Verify that your URL has changed to your own account (the code should look the same)

Clone the repository to your “local” machine (one time setup)

Cloning uses the git tool to copy down all of the files from the repository and all of the history, branches, and commits of that repository to your local machine so you can work with the files and make your own changes. This automatically adds a link back to your fork for you to send changes back up later.

- Navigate to your Fork URL if you are not already there.
- Under the “Code” button, you should see an HTTPS tab. Copy the command in the box (or press the “copy to clipboard” button)
- Open a terminal window on your local machine
- Change directories to where you would like the repository code to be located, if necessary.
- Paste the **git clone** command you copied from Github (Ctrl-Shift-V in a Linux terminal)
- If prompted for a username, enter your **Github username**
- If prompted for a password, enter your **Github Personal Access Token** you created previously. Your regular Github password for logging into the website will NOT work.
- You should see a series of messages about packing objects, etc and eventually a message that shows a new local branch was created from the remote repository.
- Configure git with your account information to make updates easier in the future:
  `git config --global user.name “<Your Name>”`
  `git config --global user.email “<Github Email>”`

Daily Example Workflow (Task branching)

In order to change things efficiently and avoid conflicts with team members on highly collaborative projects, many teams adopt a Task Branching workflow, where each new task is built on a separate branch, and then merged in via a Pull Request once reviewed / approved by the team.

- Create a new branch (choose an appropriate name for the task)
  `git branch <branchname>`
- Switch to the new branch
  `git checkout <branchname>`
  (to verify: **`git status`** or **`git branch`**)
- Add your changes to the Staging Area
  `git add .` (add current directory recursively)
  OR
  `git add <filename>` (add specific files / paths)
- Commit your changes to your local repository (does not affect Github). An editor should pop up and ask you to type in a commit message that explains why you are making this commit.
  `git commit`
  (to verify: **`git log`**)
- Push your new branch and commits to the remote repository on Github
  `git push origin <branchname>`
- In your web browser, navigate to your repository and select the branch button (center top, tree icon) to view all of your branches.
- Next to the branch name you just pushed, select Create Pull Request
- When prompted for Source and Destination, make sure your own repository is the destination repository.
- Choose an appropriate title, description, etc. for your Pull Request.
- Create the Pull Request (this may take a few moments)
- You should be automatically redirected to the new Pull Request. You can examine your code changes here, add comments, perform code reviews, or approve / decline the pull request from this page.
- Choose Merge at the bottom of the page. This merges your work back into your primary branch.

Your work is now on the primary branch on Github, but this has not been reflected on your local repository yet.

- In your terminal window on your local machine, switch back to the primary branch **`git checkout <primary>`** (usually “main” or “master”)
- Pull down the latest changes (your merge) from Github
  `git pull origin <primary>`
- OPTIONAL: Clean up your now-merged task branch
  `git branch -d <branchname>`
- Repeat this process for each new task! Practice makes perfect!

Add an “upstream” remote (one time setup)

To pull updates from an “upstream” repository, or one that you don’t have write access to, unlike your fork, you will need to set up a second remote url, and use it to merge in changes. This is common when consuming repositories from other teams or when participating in an open source project where you cannot push to the original repo.

- Navigate on Github to the original repository you forked.
- Copy the URL provided in the “Code” button
- In the terminal on your local machine, add a new remote link
  `git remote add upstream <url>`
  (to verify: **`git remote -v`**)

Update from upstream repository

Once you have added the remote repository, you can pull changes from that repository into your local copy, and then push the merged changes into your own fork.

- Use the normal daily workflow to ensure you have a clean working copy.
- Ensure you’re currently on the primary branch
  `git checkout <primary>` (usually “main” or “master”)
- Pull changes from the upstream repository and merge them into your local branch
  `git pull upstream <primary>`
- You may run into conflicts if your local repo and the upstream repo have changed the same files in the same locations. Here are some helpful commands to deal with conflicts.
  - Keep only YOUR local changes to the file in conflict (throws away upstream changes)
    `git checkout --ours <file path>`
  - Keep only UPSTREAM changes to the file in conflict (throws away your local changes)
    `git checkout --theirs <file path>`
  - Edit the file and look for conflict markers (<<<< and >>>>), and resolve the conflict manually.

    - Remember to mark each fixed file as “resolved”
      `git add <file path>`
    - Complete the merge commit
      `git commit`
- Your working directory should now be clean, and include the upstream changes merged with your local changes.
- You can now push up to your fork (usually “origin”) to ensure the upstream changes are included in your fork as well.
  `git push origin <primary>`