Git/GitHub Set-Up and Commands

Oliver Löthgren

Table of Contents

Downloading git/set up a Github account

1. Follow the instructions here: https://github.com/git-guides/install-git
   a. Check if installed ("git version" in command prompt)
   b. Update git ("git update-git-for-windows" in command prompt)
2. Sign up at www.github.com
3. Note: In the below commands {} shouldn't be typed, it's just placeholder with information inside on what should be typed out.

Full set up steps (to type in console)

Get git

1. Open command prompt and type `git version` to check if git is installed.
2. If it is not, download git via the following link.
3. If it is, make sure it is up to date by typing `git update-for-windows`

Set up local repository

4. Create folder, anywhere, that you want to be the main project folder, then open the command prompt and type `cd "path to folder"` with the quotation marks
5. Type `git init` to initialise a local git repository

Link git to github account

6. Type `git config --global user.name "your github username"`Ty
7. Type `git config --global user.email "your github email"` to link to your github email

Connect to github online repository

8. Type `git remote add origin "link to online repository"` . Note, the link to the repository is the https link from github when pressing the green Code button.

Pull from the online repository

9. To get files up to date, pull from the repository by typing `git pull origin main`
10. Now you can make changes to your local system, then update the online repository by committing -with a message- and pushing (must be done in this order), or get new changes from the online repository by pulling (should be done whenever you plan to start coding, in case a collaborator has already made changes).

Essentials (push, pull, commit, conflicts)

Now that everything has been initialized, and files can be added/changed inside of the local repository (on your specific machine) and then they can be pushed to the remote repository. The key processes in git are pull, commit, and push. Meaning:

1. PULL = Extracts new additions (pushes) from other people to the remote repository onto your local repository.
2. COMMIT = when a change to a file is made, it is important to "commit" these changes. This is a way of keeping track of changes/creating a history of old files incase something goes wrong. A message is also needed for explaining the changes.
3. PUSH = "pushes" all commits onto the remote repository, so it carries over all local changes to the remote repository.

The remote repository is the "final product" and the local repositories are just current versions that each person has on their own computer/is making changes to. When you commit/push/pull Git sorts out stuff in a smart way so that your code isn't overwritten or any weird changes are made. To prevent issues (called conflicts), follow these rules:

1. Always pull before you start working again (to get all new changes)
2. Always commit changes you've made
3. Push these changes ONLY when you are sure everything works/it doesn't mess with anything
4. Need to be careful when working on the same files with pushing/pulling – can create "conflicts". All commits are stored in case things go wrong and you can always go back to those versions.
5. Need to split work so that conflicts don't occur.

To fix conflicts: On your local repository, change conflicting files to be equal to the remote repository's version of that file. Commit this, push it, and the error should be fixed. You can get your previous code from your other commits.

Git Console Commands for working in LOCAL repository:

Note, all commands will take effect in the branch that is being worked in

1. ***git status***
    a. gives information about the local repository:
        i. which branch you are working in
        ii. whether there are any commits that have been made
        iii. if there are changes that need to be committed
2. ***git add {name of file}***
    a. adds only the file {name of file} to the "staging area" – the area for files that have changes in them to be committed. No " ", just the name.
3. ***git add .***
    a. adds all files to staging area.
4. ***git commit -m "{message}"***

    a. Commits all files in the staging environment, with a message (50 characters max) explaining the change.
5. ***git push -u origin {to branch name e.g. main}***
    a. pushes all changes in the staging environment onto the main branch of the remote repository “origin”. (More on branches in next section)
6. **`git pull origin {branch name to pull}`**
    a. pulls from the remote repository “origin” the code in the specified branch
7. `git reset –hard`
    a. removes all local changes since the last pull (so that committing is not necessary)

Branches to avoid errors

When working on several complicated aspects of a project, you can create a “branch” away from the main branch (the default and main branch for commits and such). New branches are essentially environments separate to the main branch that can be worked in to avoid errors. When the work done in a branch is completed, the branch can safely be remerged with the main branch

Create new branches:

git checkout -b {name of new branch}

Change the branch that is being worked in:

git checkout {branch name}

a. VERY IMPORTANT to be in the correct branch (typing the above when already in the branch will tell you that you’re in the branch)

Merge changes from main branch to another branch:

WHEN WORKING IN THE OTHER BRANCH (check with git checkout {branch name})

a. Type ***git merge main***

Remerge the branch with the main branch:

WHEN WORKING IN MAIN BRANCH

a. Type ***git merge {name of other branch}***

Deleting a branch:

Type ***git branch -d {name of branch}***

Seeing all branches:

Type ***git branch -a***