Stephanie Mah

Git Overview with Suggested Workflow

So you still don't understand git…

First off, why do you personally need it?
Imagine you got some code working but maybe it wasn't the best and you want to rewrite it. Rather than saving and duplicating files in the event you want to revert to your previous implementation, git allows you to revert to previous commits - this is one benefit of *version control* (it's like less cool time travelling)

And why does my team need it?
It allows each person to work independently of each other on a whole and up to date version of your code rather than working on duplicate copies and having to manually merge them together.

So what is a good strategy for working with git?

Here are a few tips:

- Master branch should always be functional, not broken - this ensures that no matter what happens with your branch, you can always start fresh with something that works
- As above, only merge a branch into master when you know it works
- Sharing a branch with another team member is ill-advised - it can happen, but wherever possible, either use separate branches or branch off of their branch
- Always make a pull request (PR) to merge your branch into master (or any other branch really), DO NOT just merge straight in with a git command - PRs will expose potential merge conflicts and will allow yourself and others to see the difference between the two branches so that you may resolve any issues before you merge.

With that said, how do you actually do the things???

`git clone project_URL` - use this to **get a local copy of a project** for the first time - you'll get a local copy of master on your machine
`git checkout -b branch_name` - use this to ***create and navigate to a new branch*** off of whatever branch you're on
`git checkout branch_name` - use this to ***navigate to an existing branch***
`git pull` - use this to ***pull any changes in the remote version*** of your branch to your local version - switching branches does not always do this automatically, always do this

after you've switched branches especially when you switch to master
`git pull origin master` - use this to ***pull changes in the master branch to your local branch*** - may result in merge conflicts! But your IDE should point any merge conflicts out to you and ask how you want to resolve them
`git add .` - use this to ***add all changed files to your upcoming commit***
`git commit -m "message"` - use this to ***bundle a set of changes together*** before you push - should be used after `git add .`
`git push -u origin branch_name` - use this ***the first time you commit changes on a new branch*** - after creating a new branch, it won't exist remotely until you run this command
`git push` - use this ***to push committed changes to the remote version of your branch*** - will error if you didn't do `git push -u origin branch_name` yet on a new branch

Suggested Project Workflow

As we typically require you to submit a branch for code review, I would suggest your workflow be as follows:

1. `git checkout master` if you're not already on master
2. `git pull` to get the most recent version of master on your local machine
3. `git checkout -b code_review_branch` to create a new branch off of master -> *Only one team member should do this!* Treat this branch as your master!
4. `git checkout code_review_branch` to navigate to the new branch (If you were the member who created the branch, you are probably already on this branch)
5. `git checkout -b stephanies_feature` to create your own branch off of code_review_branch -> *Each team member should do this!*
6. Make some changes to the code
7. `git add .` `git commit -m "new sweet feature added"` `git push -u origin stephanies_feature`
8. If your feature isn't done, make some more changes and then `git add .` `git commit -m "fixed mistake in last commit"` `git push`
9. If your feature is done, make a PR! Go to your github repo and click the "New pull request" button. Add your teammates so they can review your changes and make sure you're merging `stephanies_feature` into `code_review branch`.
10. Fix merge conflicts if there are any. You can pre-empt this step if you run `git pull origin code_review_branch` before your PR and fix merge conflicts in your IDE.
11. Wait for approval (should talk with your team on how many approvals is enough. One person? Everyone?)
12. If there are changes to be made, any new commits to your branch will automatically be added to your PR.
13. If approved, merge your PR! 🎉
14. If you're still working on code_review_branch, `git checkout code_review_branch` `git pull` and start from step 3!
15. If you're done with code_review_branch, one team member should make a PR to merge `code_review_branch` into `master`. Once it's merged, start again from step 1!

For anything else such as `git status`, `git diff`, `git revert` ask a #question or google it, there are lots of resources out there on how to use git. :)
Happy coding and may the merges be in your favour! 🎉