Workflow Overview

lesson #dev02

James L. Parry
B.C. Institute of Technology

Lesson Goals

This short lesson introduces some project workflow techniques.

Atlassian has some great tutorials, and I am using some of their images here.

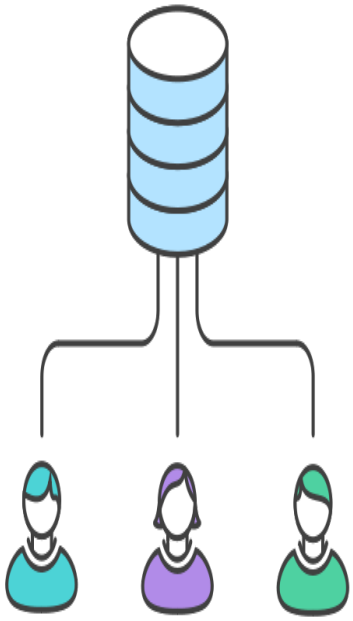
Github itself has some handy guides, on the GitHub Flow, for instance, and on forking projects.

Centralized Workflow

With a centralized workflow, all the developers on a project share a central repository.

They clone the repository locally, but they all publish changes to the shared repository.

There is no locking or ownership, so this is problematic if someone else changes a file you are working on.



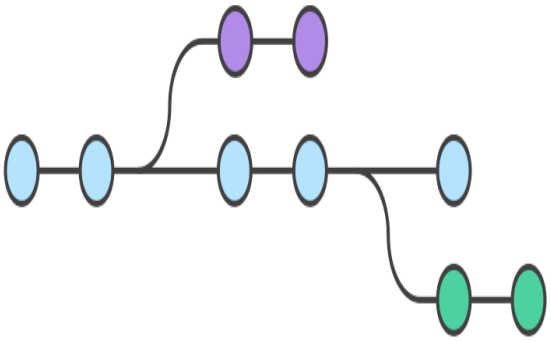
Branching Workflow

Following a branching workflow means that all feature development takes place in dedicated branches, other than "master".

Once a feature is complete, its branch would then be merged into the "master" branch.

Fixing an issue is considered a mini-feature, and would be handled in its own (short-lived) branch.

This is better than a centralized repository, but there can still be conflicts.

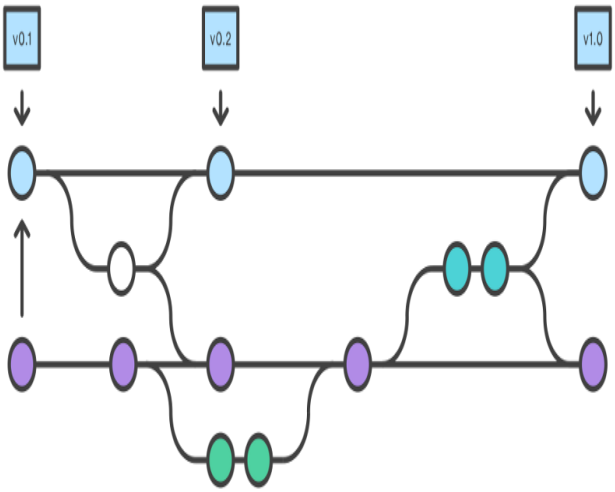


Gitflow Workflow

Gitflow workflow is a strict branching model convention.

You have two main branches, "master" and "develop". The "master" branch is always release-ready, and is usually tagged with a release version number.

All development is done on the "develop" branch, which is merged into the "master" branch to form a new release. If you only deploy from the "master" branch, conflicts don't affect it.

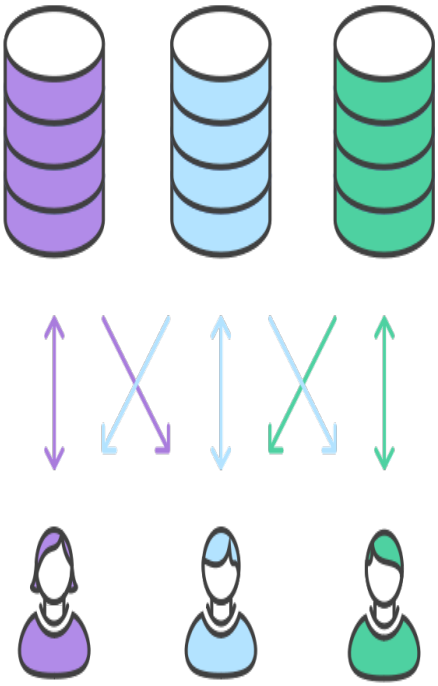


Forking Workflow

With the forking workflow, there is a main repository, and each developer "forks" it to make their own server-side repository, which is then cloned locally.

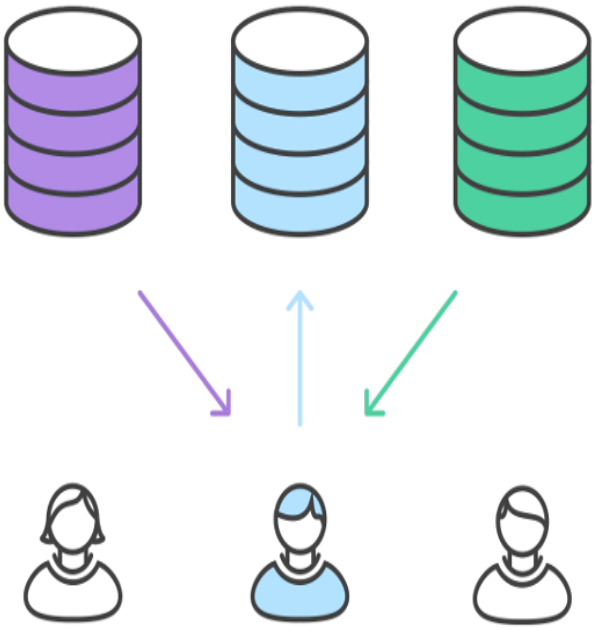
Developers push code changes to their own repository, and trigger a pull request when they are ready to merge their changes into the official repository.

Only the maintainer of the official repository can update it. They may include updates that break your repo, but that is "your" problem.



Github Workflow

1. Fork a github project
2. Clone your fork locally
3. Create a topic branch
4. Commit changes to your branch
5. Push your changes to your fork
6. Send a pull request to the original project



Make Sure You Are Signing

Many projects insist that submitted code be "signed", so they can be assured of the identity of the contributer.

The first part of this is making sure that Git is configured with your name and email. This might be doable through your IDE, or you might have to use the command prompt, shown right.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

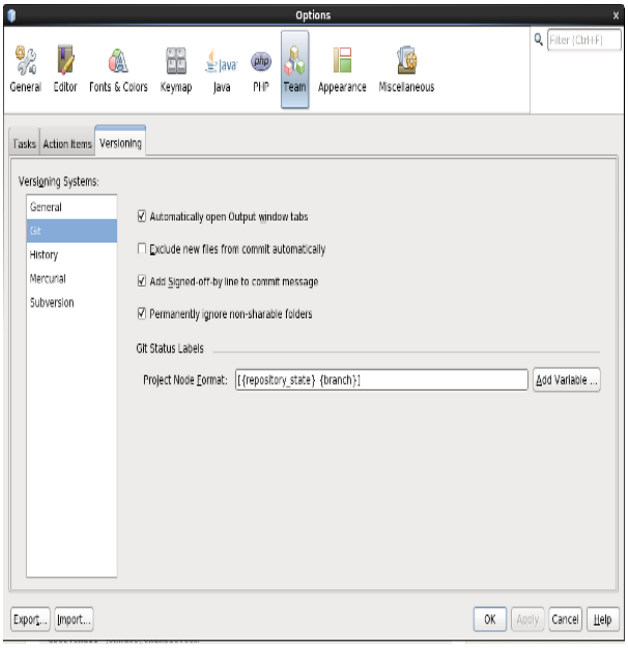
```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Tell NetBeans to Add a Signoff

In Tools/Options/Team, set the Git open to "Add Signed-off-by line to any commit message automatically.

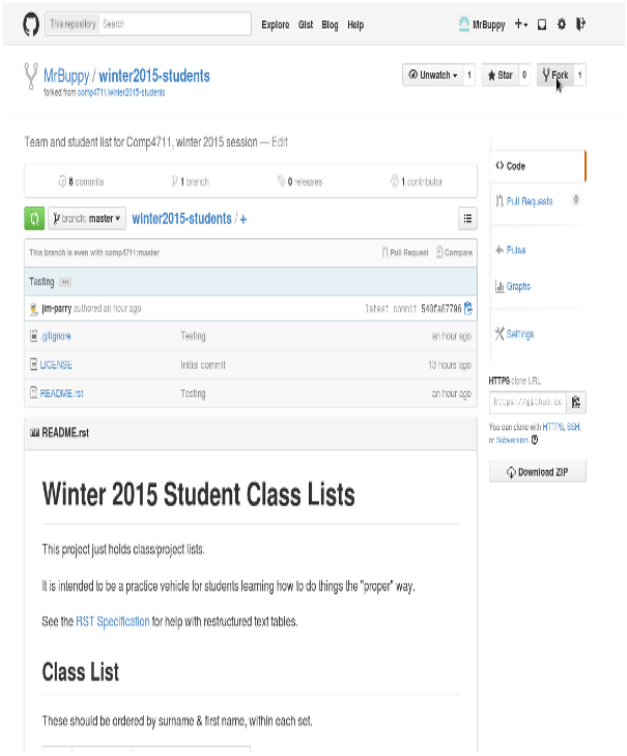
Many projects insist that submitted code be "signed", so they can be assured of the identity of the contributer.

This signoff might be acceptable for many projects, but some will insist that commits be "digitally signed". This process is more involved than you need for our purposes.



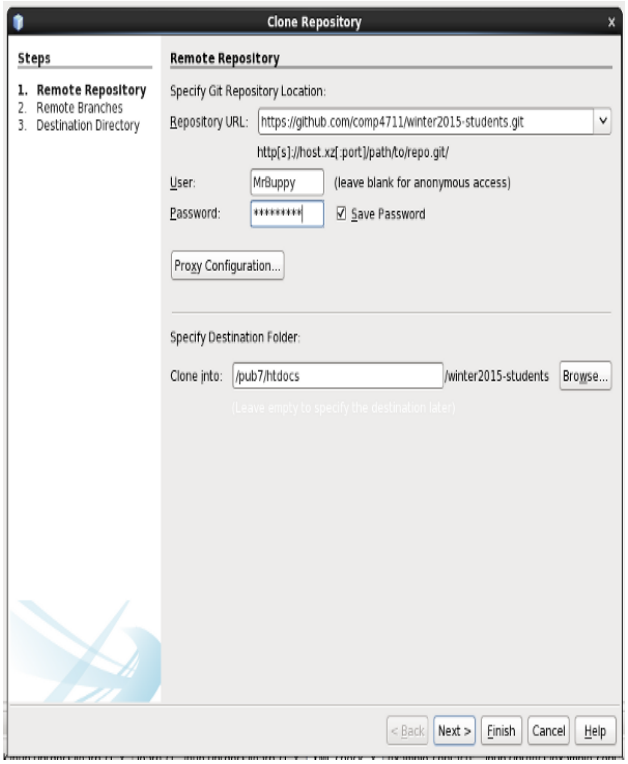
Fork Our Project

On the Github website, fork the repository you want to work with or contribute to.



Clone your fork locally

You should remember this from last week :)

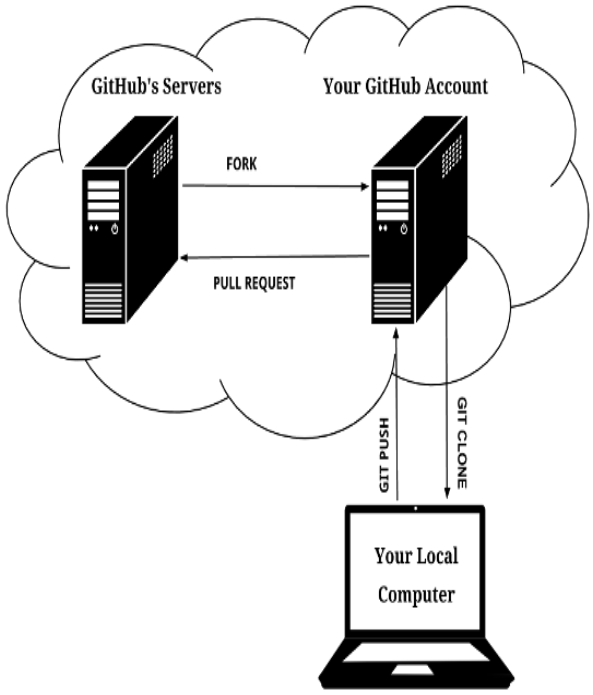


Repository aliases

You work locally with your repository clone.

Your (forked) repository is given the alias "origin".

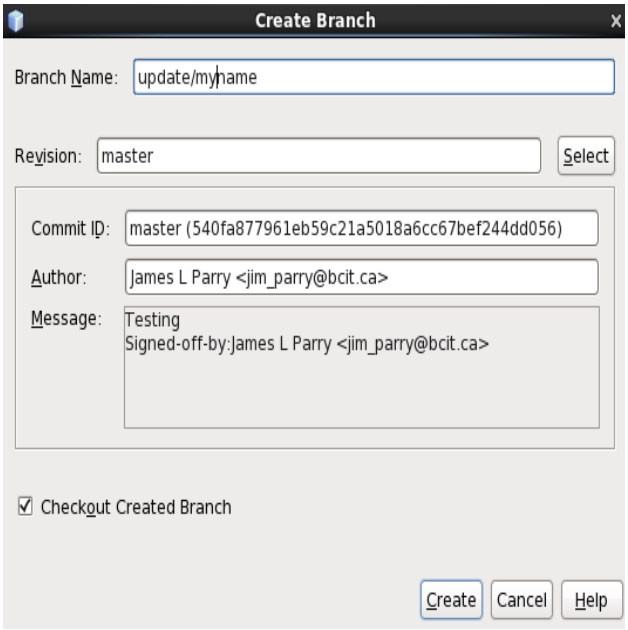
The original repository should be given the alias "upstream", in case you need to sync with it. This might be easier to do from the command line.



Create a topic branch

Create a new branch, based on "master" or "develop", as appropriate, and check it out.

If you forget the checkout, things will likely get messed up :(

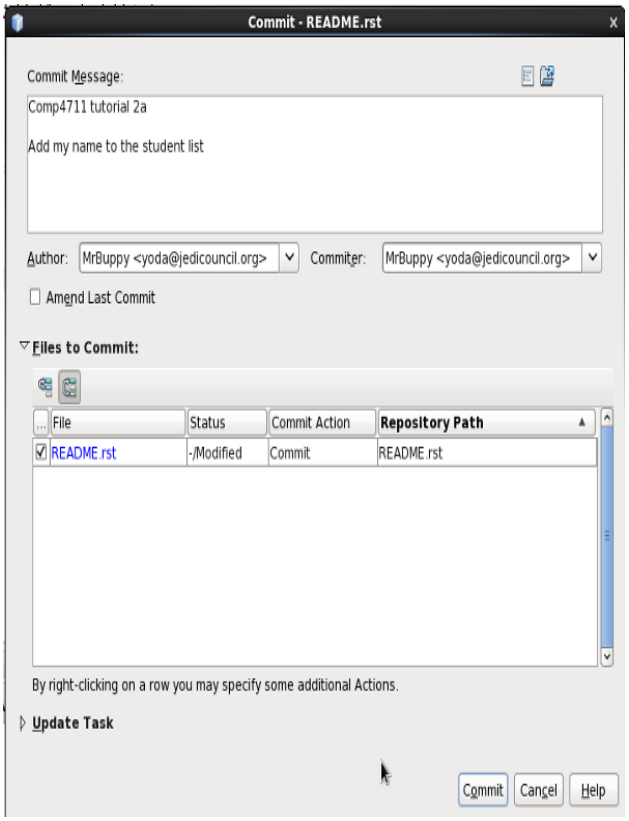


Make and then commit changes to your branch

Make whatever project changes you are responsible for.

Commit your changes to your staging area after each group of related changes.

Remember to make sure your commit is signed, and that it has an appropriate message.

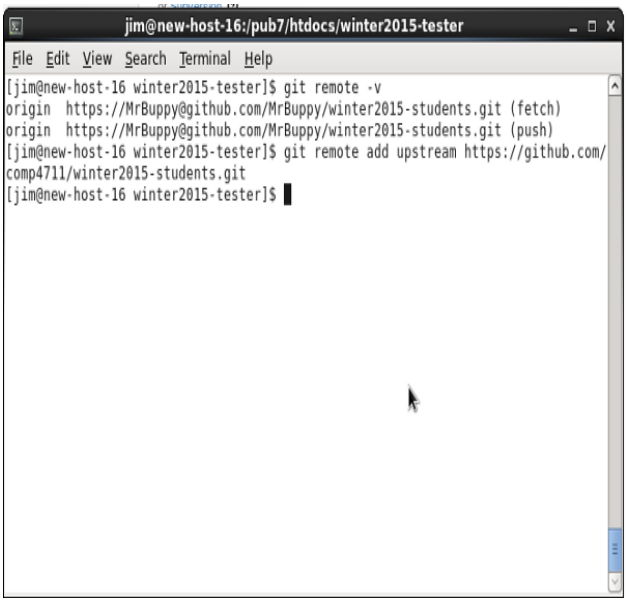


Synchronize Your Fork With the Main Repo

If any changes are made to the shared repository (for instance by a fellow team member), you will need to synchronize your repository with it. If you haven't done so already, you will need to add the remote repository, and you should call it "upstream".

You can then synchronize your repo by

1. Checkout the "master" branch
2. git pull upstream master
3. git push origin master
4. Checkout your topic branch again.

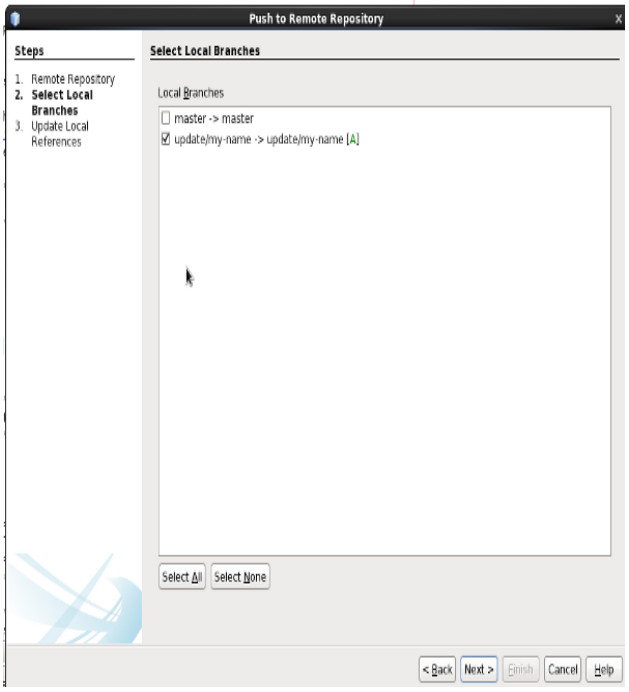


Push your changes to your fork

From your IDE, or the command line, push your changes to your server-side repository, "origin".

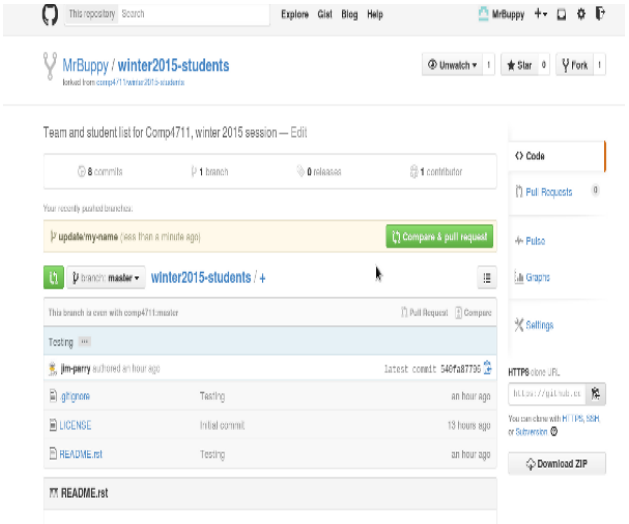
You should make sure you are in sync with the shared repository before doing this, or you run the risk of ending up with horrible conflicts.

If you get a nasty message when synchronizinging, you will need to resolve any merge conflicts before continuing.



Send a pull request to the original project

If you switch to your github repository in your browser, you will see that github is offering to "compare & pull request".



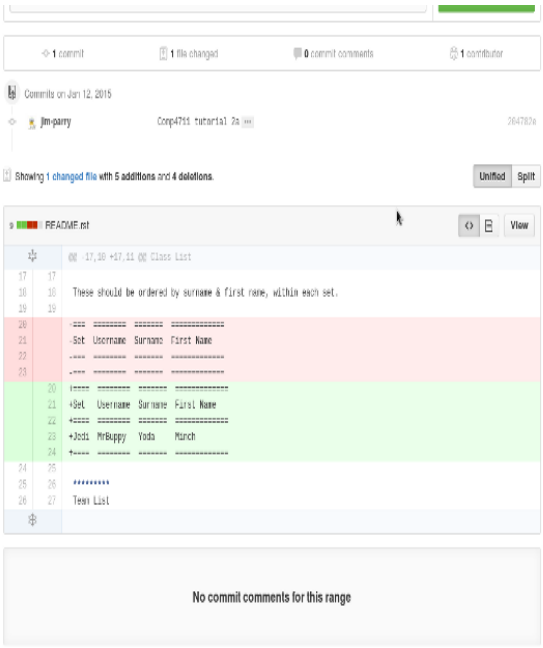
Initiate the Compare & Pull Request

Select the "Compare & pull request" link.

Near the top of the page, it should say that the pull request can be automaticaly merged. If it doesn't, you did something wrong, most likely forgetting to synhronize with the shared repository before pushing to your fork.

Scroll down to the bottom of the page displayed, and make sure that the changes are what you expect.

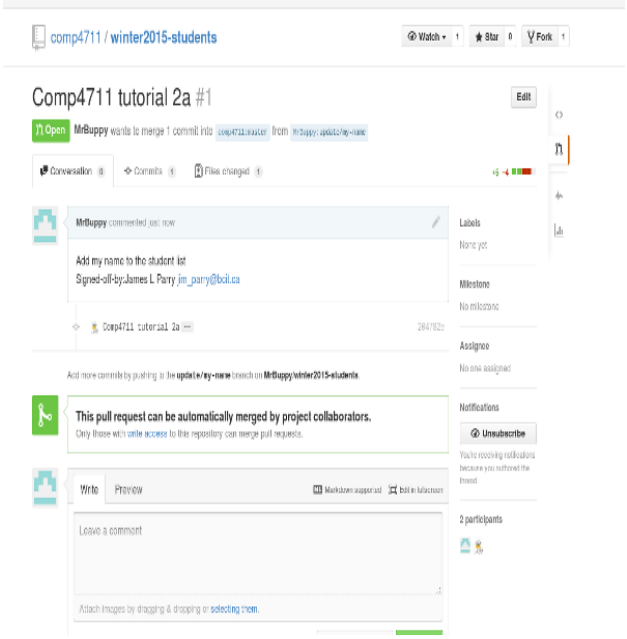
If all looks ok, click "Create pull request".



Wait For It...

At this point, you should see the main repository, with your pull request center screen.

You can comment on it, but you will have to wait for the repo maintainer to merge or reject your pull request.



Are We Done Yet?

Your pull request might get rejected for any number of reasons. You will get an email explaining the decision.

Once your pull request has been merged, resynchronize with the "upstream" repo, and you can delete the topic branch on your github repo as well as locally.

This probably sounds like a lot of work, but it will start to come naturally to you, and you will appreciate the wisdom of shared code management, and some of the steps here, once you have two or more developers sharing a repository!

Congratulations!

You have completed lesson #dev02: Workflow Overview

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: php01 PHP Crash Course

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.