Final class: Project, GUI clients, Review

Class 5 of 5

Last class

Welcome to the final class. I hope you have learned a lot about git.

If you're wondering about how long the course material will be up - you will have access to the Moodle course for a year. Heather and I will keep up our supplementary site for at least a year. And feel free to email us with questions – really, not a lot of you have so far, so it's not like we are overwhelmed with questions – please don't hesitate to ask, even if it's next September.

What we'll cover today

- GitHub pages
- The final project
- GUI clients
- Review

I'm going to go over GitHub pages – and your final project, which includes GitHub pages. We will talk about your final project – which is obviously more for you, than us – but you may as well work on it while the git material is still fresh in your mind. It's not very complicated to do at all – especially since GitHub has made stuff easier on their side in the last couple of years.

We will also spend a bit of time looking at some GUI clients to use with git – so instead of using the command line, using a program where you click buttons and menus to interact with git.

Someone in the survey said they were having a bit of trouble understanding the git concepts – so I thought it would be helpful to have a review.

GitHub Pages

- Host a website in GitHub
- Default user & project pages
- Static sites only, 1GB max
- Can use custom domains

One feature of GitHub that's really nice is that they will host small, static websites directly from your git code – this is called GitHub pages. You can have a user page and also a website for each project. The limitations are that it has to be a static site – so you can't use server-side languages like Python, PHP or Ruby – and it has to be under 1GB. You can, however, link it up to a custom domain – you'll see that be default the domain will be something like username.github.io – but it doesn't have to be.

An example of a GitHub page is our supplemental website at http://hklish01.github.io/gettingtoknowgit/index.html. I'm actually working on this on local, and pushing up to Heather's repository on the gh-pages branch – which is kind of an artifact of how long ago we put up this repository – you'll see in a bit we don't necessarily need to use this gh-pages branch anymore.

GitHub Pages: User pages

- URL will be something like github.username.io
- How to set it up:
  - Create a repository that is the same as your user name + github.io (so mine would be *katebron.github.io*)
  - Work on project locally, push back up
  - Uses master branch

Github differentiates between a user page and a project page - the main difference between these is how the URL is set up. With user, it's just github.username.io, with projects you will have something different on the end of the path (like Heather's gettingtoknowgit site). We'll actually be using a project page for the final project, even though the project acts like a user page, I don't want to assume that's what you'd like your user page to look like. You can always change the repository to be a user page if you'd like.

To work on a user page, you create a repository in GitHub, and name it the same as your user name, with github.io at the end so – username.github.io – like with other projects you'll want to clone it to your local repository and add files, and then push it back up to GitHub. I should have mentioned last week, however, that you can edit any file in GitHub – it's easy for small changes but again for most realistic development you probably want to see how it looks on local – you can't really test out your code on GitHub – at least not without making a commit.

So the last point is for the user type of GitHub page, you can use the master branch to populate your website - they have changed it in the last couple of years where you actually can have a user page attached to your repository, and also you can have a website for each project. It used to be that you had to put an index.html file in a special branch called gh-pages – now you can just put your index file right in the master branch – or if needed, in the /docs folder. For those of you new to HTML, the index file is the default file servers look for to render HTML pages – if you have an

GitHub Pages: Project page

- Project pages
  - Uses master branch, gh-pages branch or /docs on master
  - Can use a pre-made theme or work from scratch
  - Example: http://hklish01.github.io/gettingtoknowgit/

Project pages – you can have one per project in GitHub. Again, our class site is a project site of sorts, even though it's using an older set up in GitHub. You can put your files on the master branch – and if that doesn't work with your project, you can put it in a /docs folder or create a branch and name it gh-pages.

With project sites, you can choose a theme and GitHub will create the style for you. I actually don't love the styles they have – for the project I have a specific site to fork, but of course if any of these styles speak to you, go ahead and use this for your project instead.

One more thing – any user or project page you create will now automatically be with HTTPS instead of http, like our old project page – we have to switch it over but I haven't done that yet. HTTPS is more secure than regular old HTTP so it's a good switch, and you don't have to worry about changing anything yourself.

Final Project: GitHub Pages

- Fork a website and make it your own site on GitHub
  - Fork this demo: https://github.com/katebron/demo_website
    - Website: https://katebron.github.io/demo_website/
  - Tweak the code to your liking
  - Push back to your GitHub fork & visit URL
    - URL will be https://github.com/*your_user_name*/*name_of_repository*

For the final project, you're going to fork an existing website, which is my fork of another person's website. I updated it a little because I think a few things have changed since this person put it up.

So what you'll do is fork my fork, clone it to your local machine, tweak it with your name, etc, change the styles if you like – and in the README, there is an instruction for a second style that you can hook up to if you object to this pink one – add and commit your changes, and then push back to GitHub. There are more instructions in the README file – which you probably want to change as well – it's actually in markdown, not HTML – I think a lot of people find markdown easier than HTML – we have instructions on how to use markdown, but both with markdown and HTML, you'll really only have to change words and not even touch the more code-y parts of the file.

One quick note to those people who have never worked with HTML files before – you can preview how they look on your local machine by just opening the files up with a browser.

When I originally thought of this project, I thought you'd be able to practice with branches – you won't actually have to create any new branches, but since this fork is a little bit old, you will actually be on the gh-pages branch when you clone it to your local machine- just keep that in mind when you push back up to GitHub in case anything doesn't go according to plan.

Git GUI Apps

- GitHub Desktop client

- Other programs
  - Source Tree
  - Gitkraken

Heather's DEMO: 13:40 – 25:00

Review

Demo of git basics

- Git set up & configuration for your computer
- Setting up a git project: init, remotes
- Adding and committing
- GitHub: accounts, cloning

Heather's demo: 25:17 – 38:30

Forking and collaboration

- Fork = a copy of someone else's project that GitHub places in your account.
- You clone your forked copy to your local machine to work on

We talked about forks – so, forking is making a copy of a project in GitHub. The copy is placed in your GitHub account, and linked to the original project. This way you can work on this forked copy without touching the original – but if you want to contribute back to the original, you can issue a pull request through GitHub.

The general workflow of working with a fork is to fork the repository on Github, make a clone of your copy of the fork on your local machine, and push and pull to and from your own GitHub fork.

To keep the fork up to date with the original project – this is especially important if you are going to contribute back to it, and issue pull requests – you do this through your local copy and then push it back up to your GitHub fork.

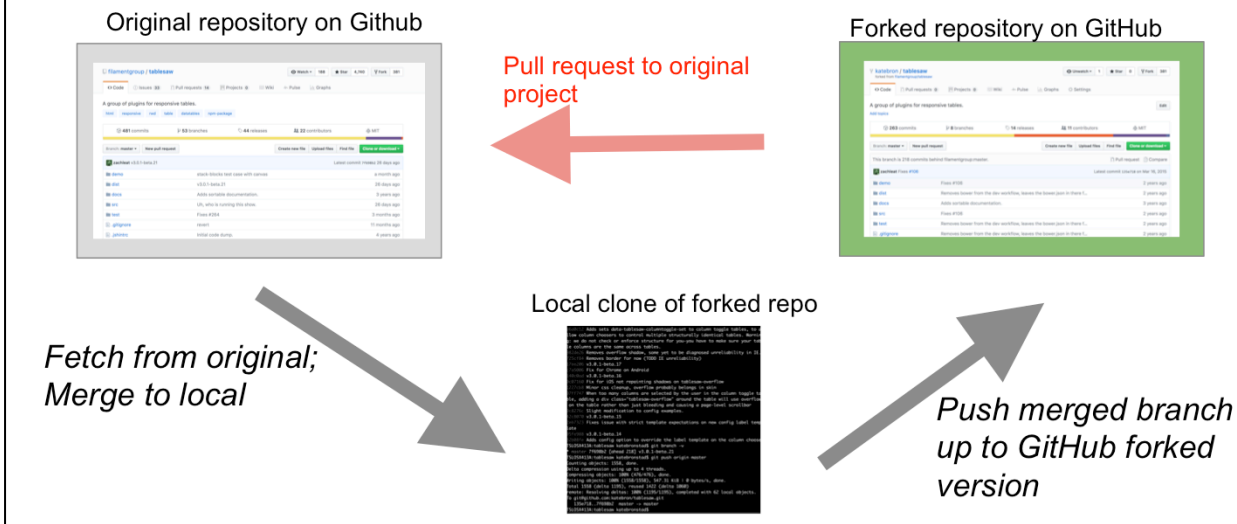
Last week we talked about how keeping a fork up to date with the original project wasn't that easy to do on GitHub itself – it turns out that it's equally not as easy on GitHub Desktop. It's doable on some of the other clients – but I did want to do a demo on the command line to give you a better idea of what it looks like.

You might want to checkout a branch to pull these in – or have a branch that has your local work - to have safer merges in case of conflicts. So if you are working on a dev branch, switch to master to pull in the changes from upstream, and then merge in

Dealing with Forks/Remotes

- GitHub client doesn't have a good way to keep in sync with upstream
  - Can use command line or Gitkraken, SourceTree
  - Command line: https://help.github.com/articles/syncing-a-fork/

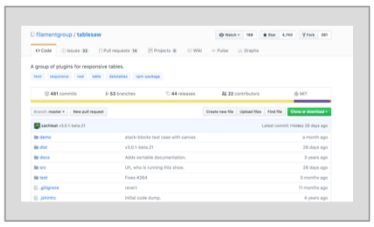
Workflow of updating original fork



I copied this slide over from last week, and added one last arrow – just to remind you, that in working with forks, you update through your local repository – pulling from the original, that's usually called upstream- merging with your local work, pushing back up to GitHub, and then if you are collaborating, then making a pull request via your GitHub fork.

Team workflow without forks

Original repository on Github



1 *Fetch from original; Merge to local*

2 *Push merged branch up to GitHub repo*

Local clone of repo

If you do have direct push access to a git repository – if it's something that you are working on with a team that trusts each other – you can just all use one GitHub repository to push and pull from. In this scenario you have to trust that anyone pushing to the GItHub repo knows what they are doing. For example, the Git repo that I use for the git pages site I showed you earlier – the one used for this class – is a repository on Heather's GitHub account, but Heather made me a team member, so I have direct access.

Working with branches

Use branches for new development and testing code from collaborators

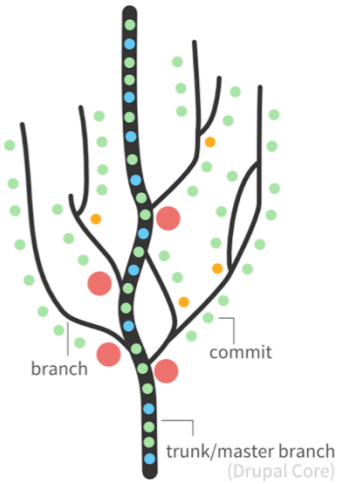


Image from https://www.drupal.org/node/991716

A branch is a copy of the project within your repository.
Look at the citygram repository (https://github.com/codeforamerica/citygram) and the different branches. Mostly they look the same, but you can use these different branches to do more experimental work. The day the person checked out 208-regional coverage from master – it looked exactly like master but has perhaps diverged in different files. If we like what we see in this branch, we can take the work and put it into the master branch. Or, we can even create a branch off of 208-regional coverage that will look exactly like 208-regional-coverage, but we can work off that branch to do even more experimental things.

Checkout

- Move to a branch or commit by check out
- You can also check out individual files from another branch or commit

You move between branches by checking them out – at least on the command line. On the GUIs you just click them.

You can also use checkout a singular file from another branch or commit, if you want to undo something you've made a mess of, or for some reason you just want to bring in one file from another branch but not others. In other words, you don't want to merge the whole branch in.

HEAD

- HEAD is pointer to the most recent commit
- Can use HEAD as a reference and refer to n away from HEAD
    - 2 commits from most recent is HEAD~2

Merging

Git combines branches by merging the code.

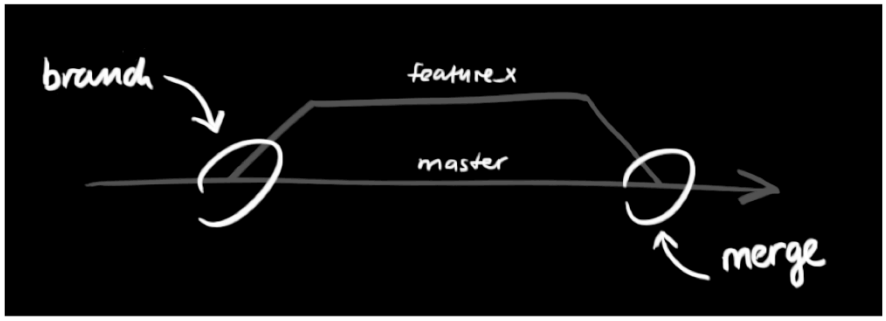


Image from http://rogerdudler.github.io/git-guide/

To combine code from two different branches, you merge them together. Merging also happens when you push and pull from GitHub or other remote.

Merging Branches in a GUI

- Click on branch you want to merge to
  - Go to “Branch” -> “Merge into current branch”
  - Brings up list of branches. Choose branch you want to merge in.

Merge conflicts

- GitHub Desktop client will show you the conflicts but you will have to open them up in a text editor

- GitKraken
  - Pro offers in-app editing
  - Can also set Preferences to default “FileMerge” program

Topics to look into

- Rebasing
- Working with tags
- Git stash
- Git cherry-pick
- Comparison of git with other types of version control

There are things that we don't have time to cover in this class, but may be helpful for you in the future. A lot of them you might not see the need for unless you are using git a lot, or on a big team of people. If you are feeling overwhelmed with understanding the basics of git, think of these as things to know about only as needed, but it's good to have a general sense of what these terms are since you will seem them referred to.

Rebasing – a different way to do a merge. If you do a rebase vs. a merge to combine branches, the code will look the same but your commit history will look different – rebasing will leave a "cleaner" history because git rewrites commits for one branch into new commits on the other.

Tags – ways to sort make a note on a commit. You can then refer to this commit by the tag name instead of the hash or the position away from HEAD. Tagging is especially important if you working with a workflow like git-flow. Each time you merge in a feature branch into the master or production branch, you'll tag it as a version number, like calling a tag "v1.0"

Stash – use the git stash command when you are in the middle of something and want to save the work but aren't ready to commit it yet. This could happen if you need to merge in code from upstream/another branch to do work on the project that isn't exactly related to what you are doing now. Git stash saves the work away, and then you run the git stash apply command when you are ready to bring it back.

Next steps…

- LITA survey
- Email us with questions
- Final project:
  http://hklish01.github.io/gettingtoknowgit/class5.html
- Resources:
  http://hklish01.github.io/gettingtoknowgit/resources.html

Someone mentioned wanting to know how to get help with the command line after class – we have some links up on our supplementary website – you can get to it from the “Resources” link in moodle. There are some cheat sheets and quick tutorials.

A note that I forgot to mention in class: you can fork this repository (https://github.com/hklish01/gettingtoknowgit) & issue a pull request if you have a resource to add to our resources page.