Git & GitHub

Roger Guldbrandsen

Software Engineer
@ Bugsnag

Bath-ML
Co-organiser

@kinbiko

XKCD comic rings true

If you haven’t installed git, please do so now!

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks

Start off with high level overview
Then Git theory
Then do hands on with the fundamentals of git
Then explore GitHub and the theory on how to work in a team using GitHub
Then do this in practice
Then close off with some final remarks

There will be many concepts introduced tonight. As such, questions are welcome at any time.
Please keep discussions of more advanced topics to a minimum, to avoid TMI and maximise hands-on time.

Why use Git and GitHub

- FINAL/DRAFT/v1.2 🚫
- Work is shared (distributed)
- Create a portfolio

No need for ridiculous and unprofessional filenames
-This is especially important for programmers where file names matter!
Distributed means colleagues and GitHub also have all your files, and their history, in its entirety
Portfolio to show off/be discovered by recruiters

Difference between Git and GitHub

- Git
  - Versioning your files
  - Command-line program 🤓
- GitHub
  - Collaboration 👩‍🎨🙋‍♂️
  - Free web service (github.com)

Git is the actual tool that does the magic.
Created by Linus Torvalds
Github is the web service that centralises much of today’s open source projects

Git: Versioning

- Multiple versions, one file
- Go back to any version
- Any file type 👌
- Plain text files 😍

Example: document that explains a process at work
The process doesn’t work well, and your manager wants to know what changed in the last 3 months to make it so bad
You run one git command and you can read exactly what the process used to be, and even see the difference between the two versions

Binary files are files that make no sense when you open them in a text editor (not MS word!)
Plain text files include configuration files and source code

Can always see who/when/why a file changed.
With plain text, can see detailed per-line details and differences

GitHub: Collaboration

- Explore other projects
- Ask open source maintainers for help*
- Work on the same files in parallel
- Review each other's work
- Project management tools and issue tracking

In particular if you work with code, you can see what’s going on with your favourite libraries and even ask questions from the developers at, say, Google, Facebook and Microsoft!
Please don’t abuse this power!
That aside, it’s great for:
- working on the same files at the same time
- reviewing work,
- pretty decent project management tools.

All 100% free (when done publicly - private repositories are cheap, but not free).

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks

Can use git through the command line and using a GUI

Will talk mainly about using the command line:
- That’s what you will find questions and answers for on stackoverflow
- That will introduce the correct lingo
- That’s what I personally use and is most comfortable (debugging) with

It’s Git, innit?

- `git init`
- Directory => repository
- Used for new projects only
- GitHub makes this a lot easier

Running `git init` inside a directory turns it into a repository, where git commands work.
Be careful not to run `git init` inside a folder that’s already within a repository. This is an advanced git technique, called ‘submodules’, which causes more pain than convenience.

Basic workflow

- `add:` marks the files you want for your next version
- `commit:` creates a version for the added files

`add` puts stuff in a staging area
`commit` is the final confirmation of what the version comprises

The order matters: `add` first, then `commit`

`git add`

- `git add <file1> <file2> …`
- `git add .`
- Adding files partially is possible (LAAEFTR), but prefer smaller changes

`git commit`

- `git commit -m “<what changed?>”`
- Remember the “`-m`”, or learn Vim basics.
- Prefer many commits to large commits

If you accidentally leave off the `-m “message”` bit, your default text editor will open. Often (esp. Macs) this is Vim.
To exit vim:
-Press ESC a few times
-Press colon, q, exclamation point, enter
-Your message is now aborted

Commit messages

Commit messages require discipline
Generally short messages - as your commits should be small ;)
But complete descriptions are better than being consistently short

git status

- **staged**
- **not staged**
- **New files (untracked)**

most used command.
Green files are files containing changes you've `git add`ed
Red files are files containing changes that you've not (yet) added.
New files also show up as red, but in a different section to the changed files, as these were previously unknown to Git.

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks

Learn by doing #1

- Create a folder called `repos`
- Create a folder inside `repos` called `my-first-repo`
- `cd my-first-repo`
- `git init`
- Create and edit a `README.md` file
- `git add` & `git commit` until confident
- `git status` is your friend

Let’s play around with the commands we’ve learned
-init
-add
-commit
-status

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks


github
SOCIAL CODING
BUILD SOFTWARE BETTER
TOGETHER

Create a GitHub account

- github.com/join
- Tell me your username

Registering should be intuitive.
Put all users in the git-learners team on Bath-ML's GitHub.

Explore

- Your favourite OSS projects are probably here. ⭐
- Your favourite developers are probably here. 🤓

Check out the “explore” heading
Find and star some projects you’re interested in
Find and follow your favourite developers, or follow each other

Public repos for everyone

- github.com/new
- Use `kebab-cased` names
- `README.md` and `LICENSE`
- Code 💛s `.gitignore`

kebab-case convention. easier to type than using upper case characters or underscores.
GitHub also suggests some pretty schweet names.
README is automatically rendered when viewing projects
LICENSE for legal purposes. There's a nice webpage for helping you choose
gitignores help you avoid mess when writing code by ignoring certain files

`learning-github`

- `clone`: get repo from somewhere else (GitHub)
- Only owners can push ☝️
- Anyone can clone & pull 👌

Let’s discuss how individuals can contribute to GitHub projects
Cloning = download repository (from GitHub) - only need to do once.
So what’s this pushing and pulling stuff?

Push/Pull

- `git push` puts local work on Github
- `git pull` puts Github work on your local machine

Cannot push to somewhere that has a different version history to yours. I.e. if GitHub has some commits you don’t have, you first have to pull, before you can push.

Forks

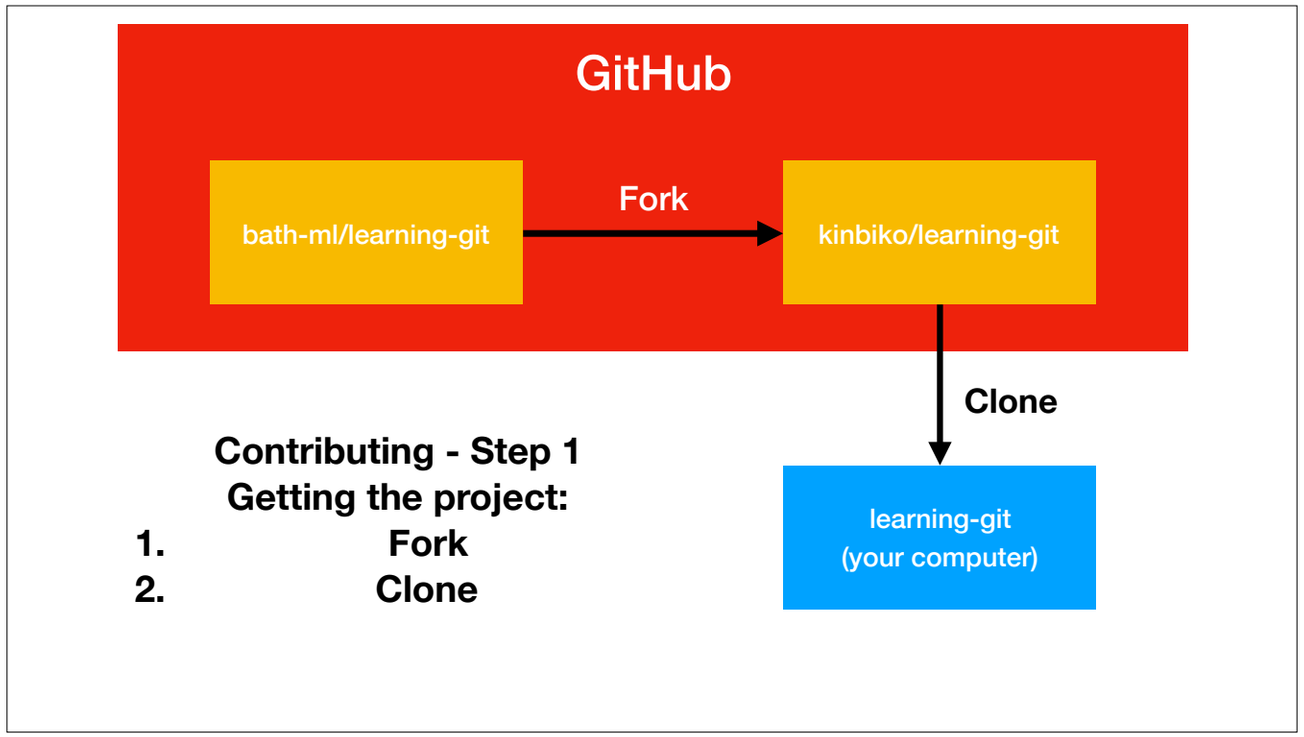
- Forks copy a project to your GitHub account
- The fork isYOURS to push and commit to
- Use the fork to contribute to other projects with pull requests

So how can we contribute to other project’s repositories if we cannot push? Solution: forks
Can think of a fork as a clone between GitHub accounts

Pull Requests (PRs)

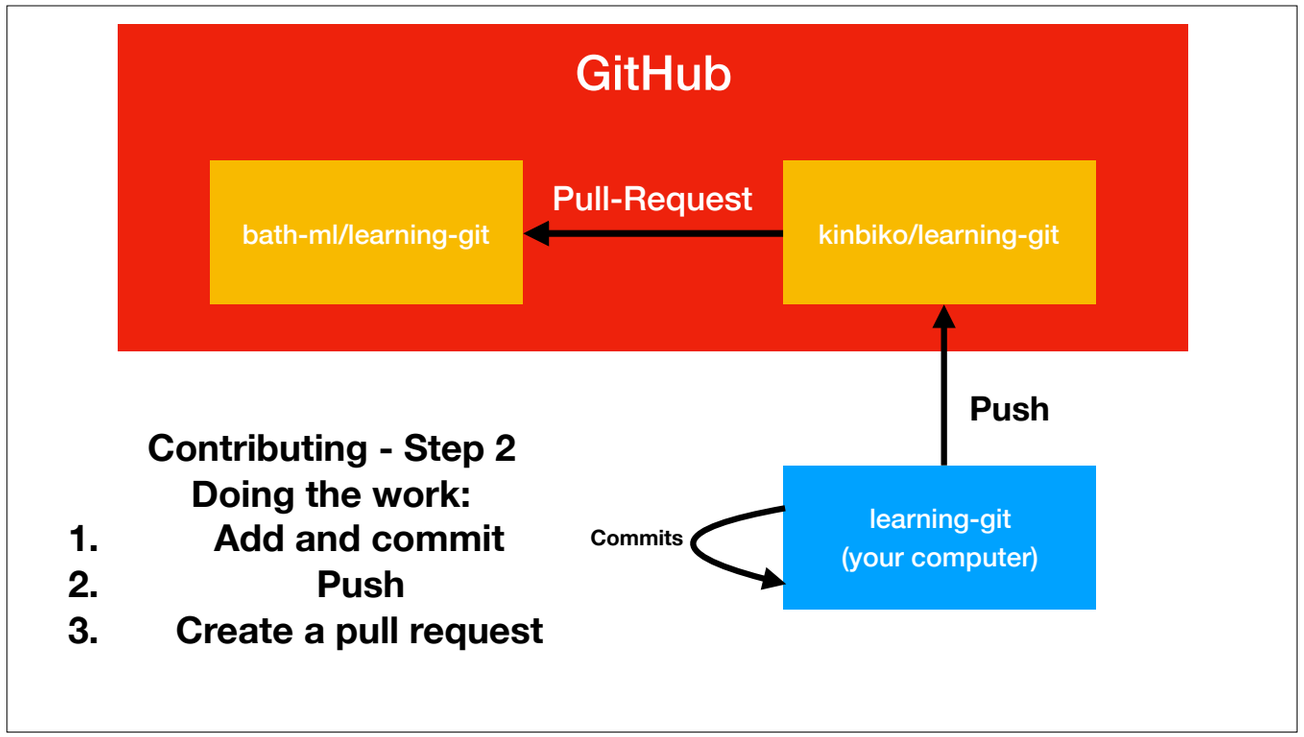
- “Please pull my changes”
- Read the CONTRIBUTING.md before raising a PR
- Add a description, assign reviewers, look over your code in the diff view

Request that another repo pulls the commits you’ve made
Primary form of review - can see a detailed view of what’s changed
CONTRIBUTING.md varies per project. Follow these instructions to ensure you don’t waste maintainer’s time
Don’t make big PRs, at least not without discussing with the maintainers first. Big PRs require a lot of time to review, and increases risk.
May not have permissions to add reviewers, reviewers may have to be added by the maintainer/team responsible for the repo



Contributing - Step 1
Getting the project:

1. **Fork**
2. **Clone**


GitHub
bath-ml/learning-git
Pull-Request
kinbiko/learning-git
Push
Contributing - Step 2
Doing the work:
1. Add and commit
2. Push
3. Create a pull request
Commits
learning-git
(your computer)

Working in teams

- 90% of the way there!
- What about updates to the original repo? 🤔
- Solve with branches!

What happens if the ‘original’ repo has had new changes that you need?
You can create a PR from the ‘original’ repo to your own, where you are the only one that can and should review. In general, just merge this. However: you’ll soon run into a maintenance nightmare if this happens often. (You’ll spend more time fixing conflicts than doing actual work)
Better solution is using branches

Branches

- Easy to create in GitHub
- `master` is holy 👼
- One branch per task

A branch is an isolated copy of your history, independent of other branches.
In teams we generally don’t ever commit directly to master, only create and merge PRs to master.
One branch per task, don’t be tempted to have one branch per person.

Checking out branches

- Create branch in GitHub
- `git pull` to update
- `git checkout <branch-name>`

Click on “branch: master” in github, and start typing the new branch name. Again, use kebab-case

Get new commits by merging

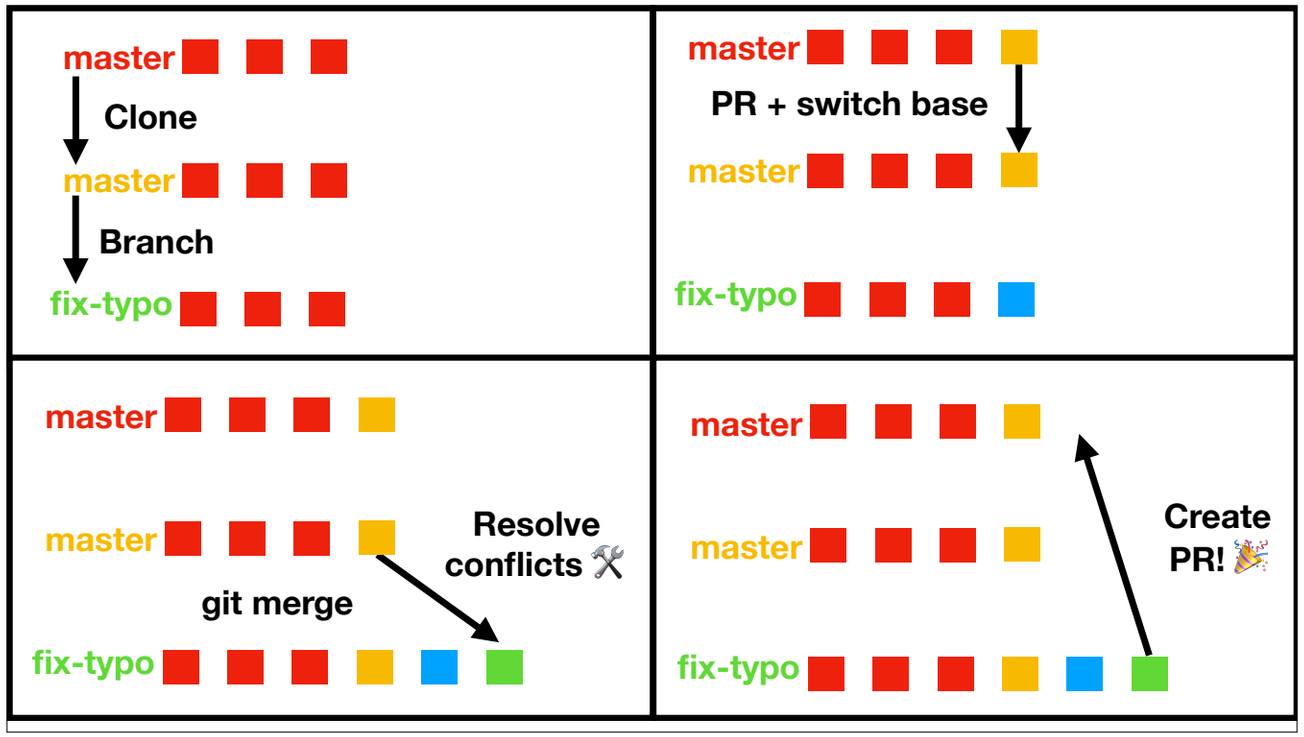
- `git pull` to update
- `git checkout my-branch`
- To update:
  - `git pull`
  - `git merge origin/master` (usually)
- (Psst! `origin == GitHub`)

If you need new stuff from master (that you've introduced through the self-reviewed PR trick mentioned earlier), do this to update:
Pulling will sync your local machine with GitHub
Merge origin/master. The 'origin' bit here is important, as your local master branch may not have the most recent changes yet.
If you ever see the word 'origin' it generally means GitHub

Merge conflicts

- When two commits collide
- Creates new commit
- ```
  <<<<<<<
  <one commit's contents>
  =======
  <the other commit's contents>
  >>>>>>>
  ```

Binary files cannot be fixed easily. Must be corrected manually :(. Food for thought: What does this mean for adding binary files to git in the first place?
When Git thinks you've changed the same files it'll try and mark what the different versions are, separated by a =======. There may be many of these. Use Git status to see which files are affected, and use a text search for '=======' etc. in your text editor for all changes.
Keep calm and consider what the right version is. Usually it's a combination of both
  ```



So what's the 'correct'* solution to handling updates on the 'original' repo?
RED = original commits, before forking and cloning.
YELLOW = Changes on the 'original' master branch.
BLUE = Your new change
GREEN = Merge commit
Step one, clone and create a branch for the task
Step two, make sure you get the most recent changes by creating a quick PR from the other repo and review and merge it yourself
Step three, merge master into your task branch. This allows you to resolve the conflicts without polluting your master branch
Step four, create a PR from your task branch to the other repo's master branch

*this is one solution. There are many other philosophies on how to do this. IMO this is the easiest that provides consistency.

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks

Learn by doing #2

- Fork `bath-ml/learning-git`
- Read and understand the CONTRIBUTING file.
- Answer (using PRs) the questions in `README.md`
- Review each other's PRs

Note: During the presentation the 'create an issue for answering the question' bit was skipped, as we were short on time due to WiFi problems.

Outline

- High Level Overview
- Git
- Learn by doing #1
- GitHub
- Learn by doing #2
- Closing remarks

Maybe don’t use Git for…

- Large amounts of data
- Binary files that changes a lot
- Secrets/tokens/api keys

Git is also used for

- CI/CD
- Hiring process
- Dotfiles

GUIs for Git

- Git Kraken
- Sourcetree
- GitHub Desktop

I personally only use the command line these days, but when I started using git I used the GUI in my IDE at the time (Eclipse - for Java development only - and highly discourage you from using this, especially if you only want a Git GUI)
If I were to use one, it'd be Git Kraken - but beware of the licensing for this software. If using for commercial use you may have to pay.
Sourcetree is the best, always free option.
GitHub desktop is created and maintained by GitHub, and has some GitHub specific goodies.

Next steps:

- GitHub bathroom tiling
- CodeHub Bristol
- Add me on social media (I'll endorse you for Git on LinkedIn)
- Contribute to Bath-ML. Blog posts and PRs welcome!

https://twitter.com/kinbiko
https://github.com/kinbiko
https://www.linkedin.com/in/roger-guldbrandsen-18065943/

If you'd like to write a blog post about what you've learned tonight, hit me up on Slack