Getting Started at Hackathons

Track 1: Gitting Started

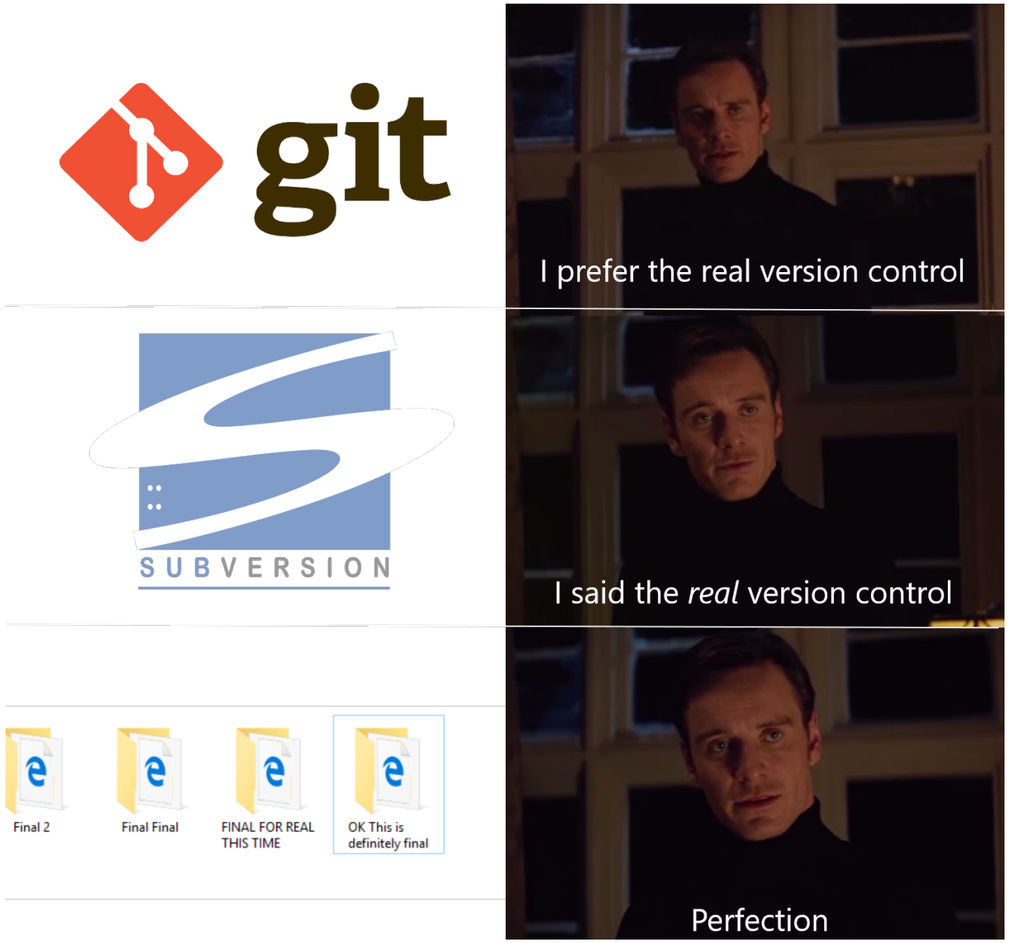
Hi, I'm **Eric Jiang** 👋

- Currently, the Project Lead for monPlan
- Co-founded GeckoDM and MARIE.js
- Co-founded and Pitched FutureYou to SMC, now spun that off as a seperate project
- @lorderikir
- https://lorderikir.me
- eric.jiang@monash.edu
- github.com/lorderikir

So, I love coding 👨‍💻 and I love working in teams 👨‍👩‍👧

But what if there was a way that I good remember how the code look liked throughout its stage, for example if something went wrong and I want to go back to a previous version?

First of all, what is git?

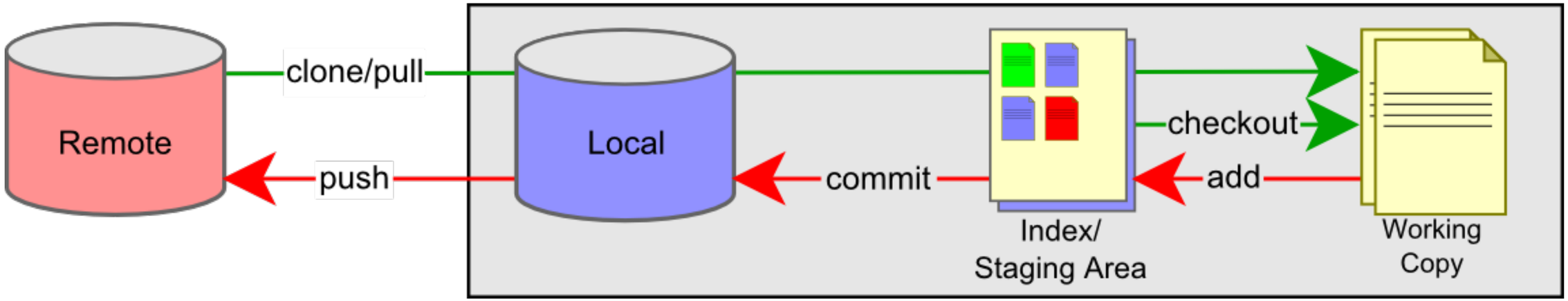


Git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people

— Git SCM Website

How Git Works



Some Basic Commands

Command	Description
`git clone`	Clones a repository locally
`git add`	Stages changes to file(s) for a commit
`git commit`	Creates a commit (set of changes)
`git push`	Push changes to the hosted repo

Using Git within teams

Well, working with teams 👨‍👩‍👧 may be hard. There are generally two ways you can work off a repository.

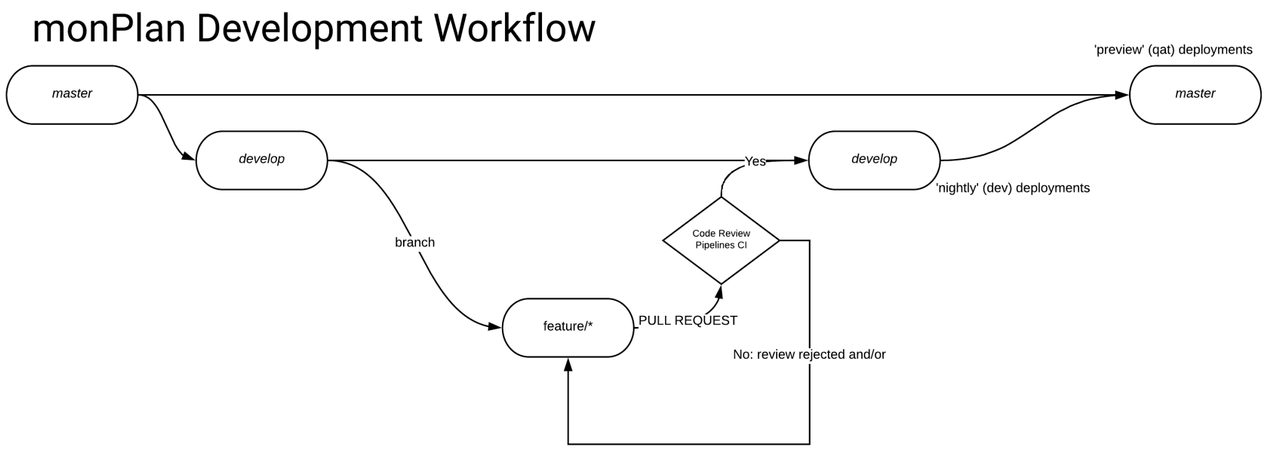
- Using Branches
- Using Forks

Option 1: Use Branches 🌳 for Versioning Control

1. Make a branch with the feature name or your own username
2. Every time you commit and push up
3. Make a Pull Request
4. Merge the pull request

One of the best workflows is known as *GitFlow*

GitFlow - Used with monPlan Git Workflow



- **master**: branch is the key branch, everytime for release
- **develop**: *unstable*, most of the PRs should go here
- **'feature/*'**, **'fix/*'**, etc.: are 'for purpose' branches, these branches are for development
- **deploy** (not shown), is for **manual** deployments to prod

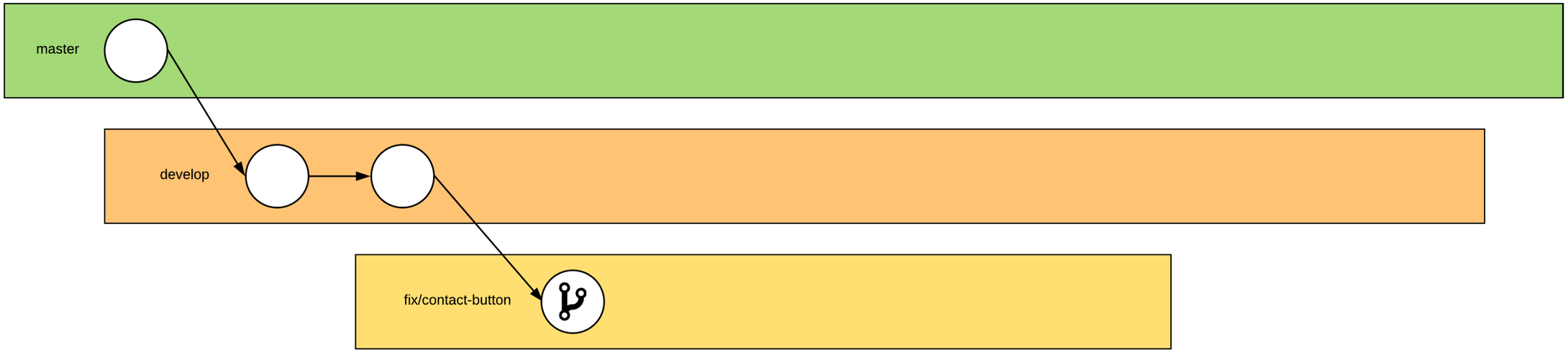
*This slide has been adapted from my **CI-CD talk***

So we know that development is done incrementally

Imagine we using Git within our practices

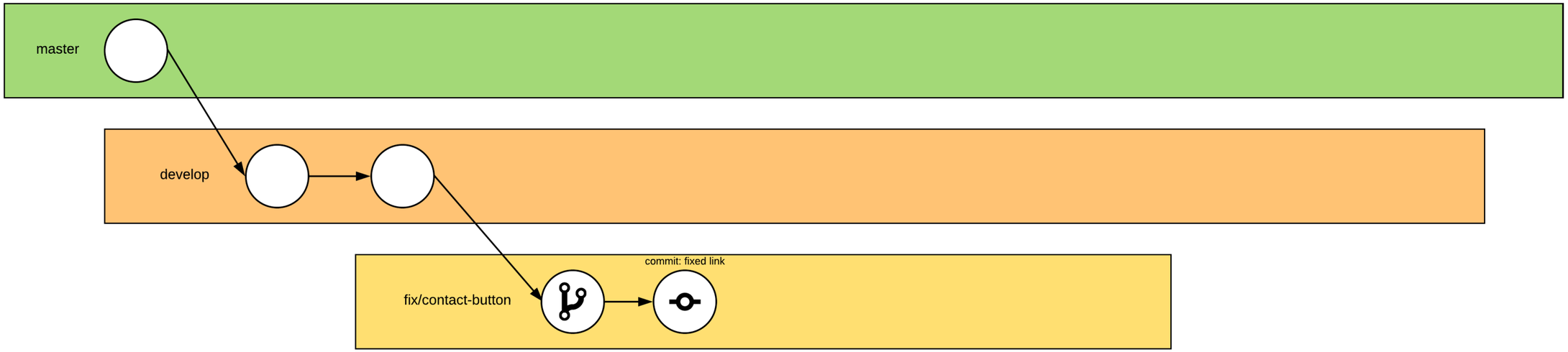
And one of my team mates, Nicholas has found a bug within one of our buttons.

So, he creates a new branch to fix the bug



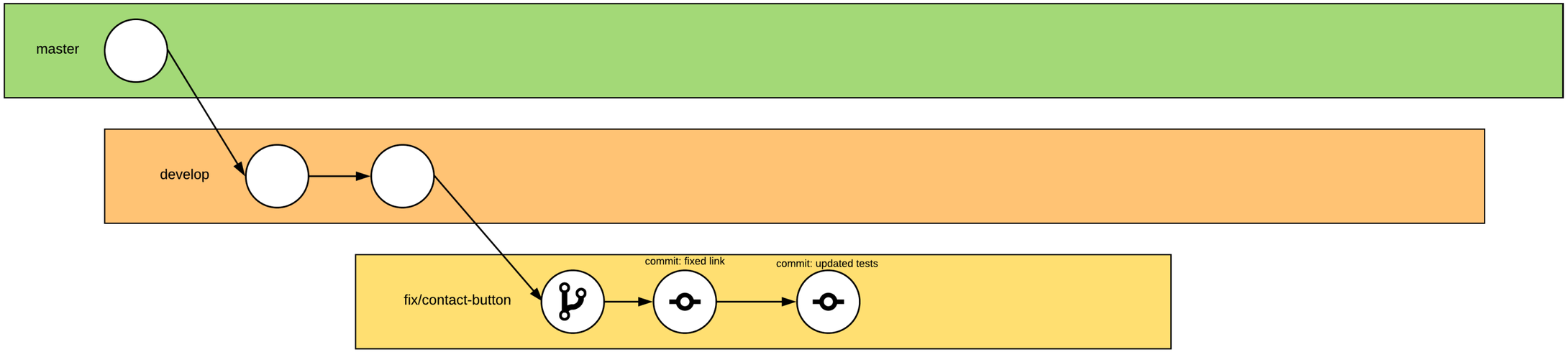

```
# update our develop branch
git checkout develop
git pull
# we create a new branch
git branch fix/contact-button
# we make the new branch the new working branch
git checkout fix/contact-button
```

He fixes the code and stages the change in commits



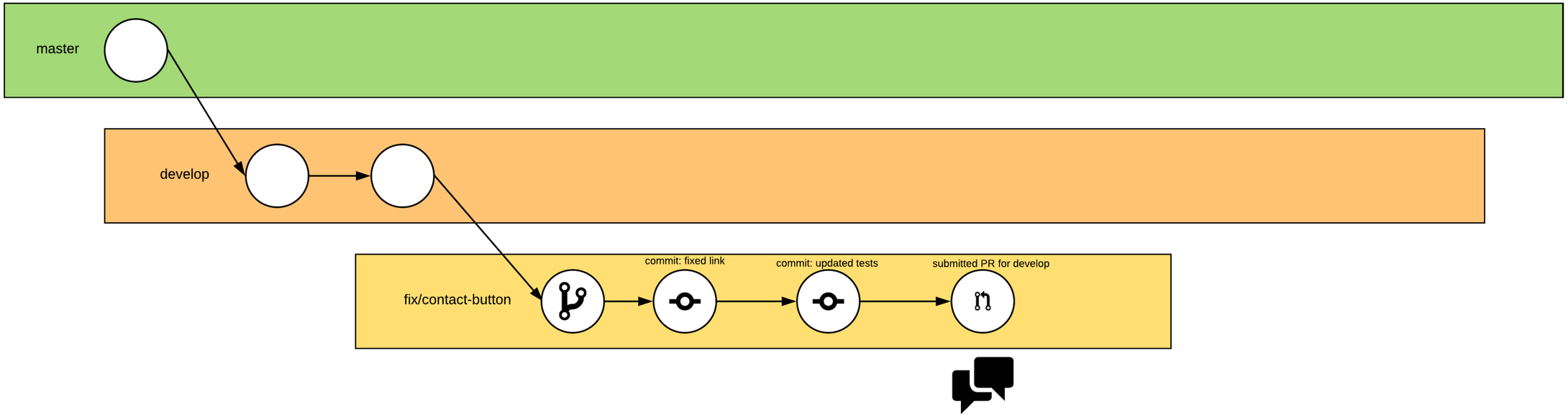

```
git add .
git commit -m "new commit"
git push
```

He fixes the code and stages the change in commits



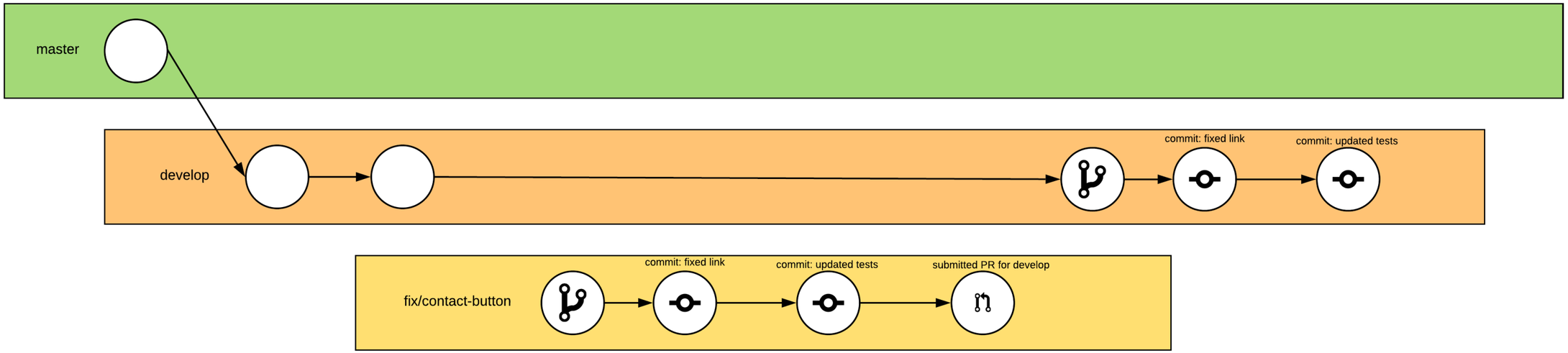

```
git add .
git commit -m "new commit"
git push
```

He then makes a PR into my develop or master branch



Where we discuss his proposed changes

We then merge the Changes



This would also work for...

- Upgrades to the codebase
- Refactoring our legacy code
- Upgrading frameworks to newer versions

Unfortunately we won't go into fixing merge conflicts in this talk

Why is using GitFlow important?

- We seperate production code and our 'work-in-progress' (WIP) code.
- We have a clearer understanding of what each developer is working on
- We can branch off WIP branches and merge changes in
- Relatively easier (not always) to fix merge conflicts
- Some CI/CD tools only run off branches (not PRs)
- We can set our CI/CD to deployment so that it can deploy off branches (i.e. `develop` to *dev*, `master` to *staging* or *qat* and `deploy` to *prod*)

Option 2: Using Forks 🍴 for Versioning Control

The best way to image a fork, is image a copy of the main repository that you own that you can *pull*, *merge* and apply changes to.

(We won't go into much detail here.)

Key notes 📝

- Version Control over Development is really important as it helps keep 'backups' and you can see the changes
  - You can always see who pushed out the broken code with `git blame` 😈
- Git is always useful as you can always revert broken code or changes
- Branching and forking is basically the same,
  - when working we typically use branches over forks as we can solve merge conflicts more easily (and locally)

Please DO NOT ever `git push --force`

Key things to look 🔭 out for.

- Merge conflicts are always the hardest part
- Be careful of `git merge` and `git rebase` commands. Always `merge` don't `rebase`
  - This is because rebase always applies your changes last (assumes you are always correct)
  - When merging between branches and fixing conflicts always work with a team-mate

Questions?

🤔 🎤 📣

Goodbye 👋

Track 2: Firebase + ReactJS for Hacks coming soon