匠人学院™
jiangren.com.au

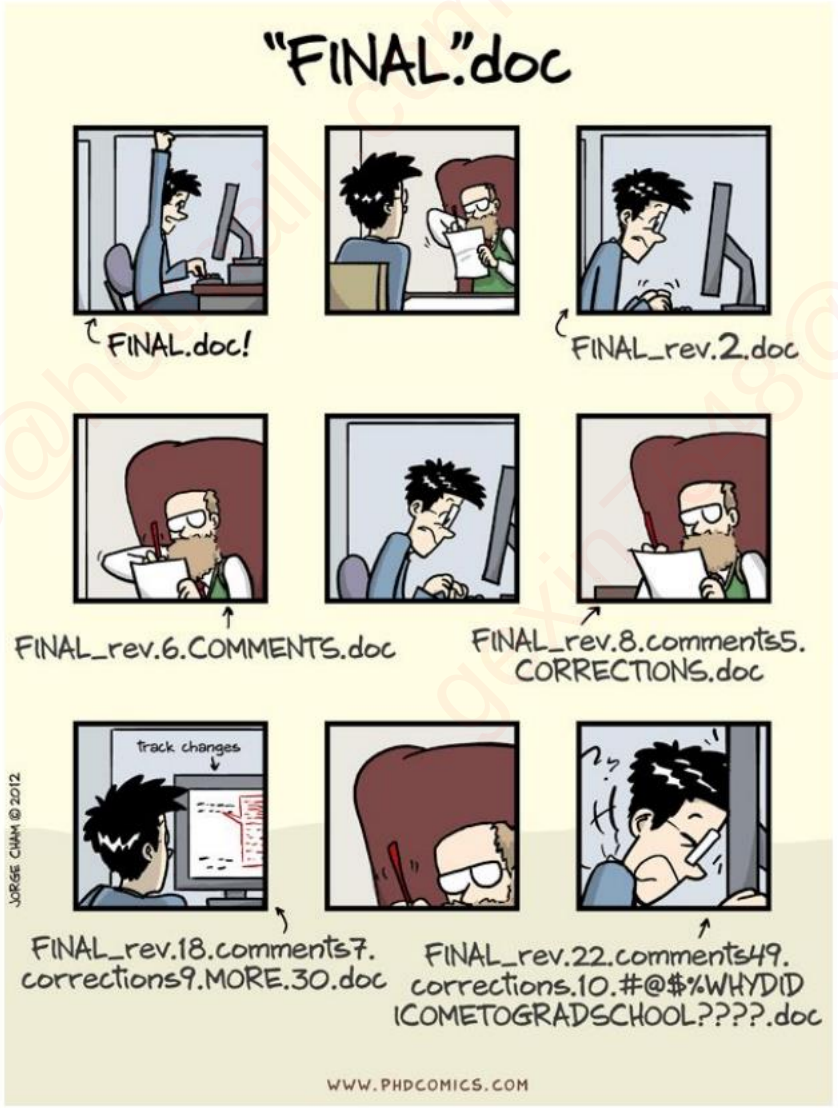
# Git Basics with Practices

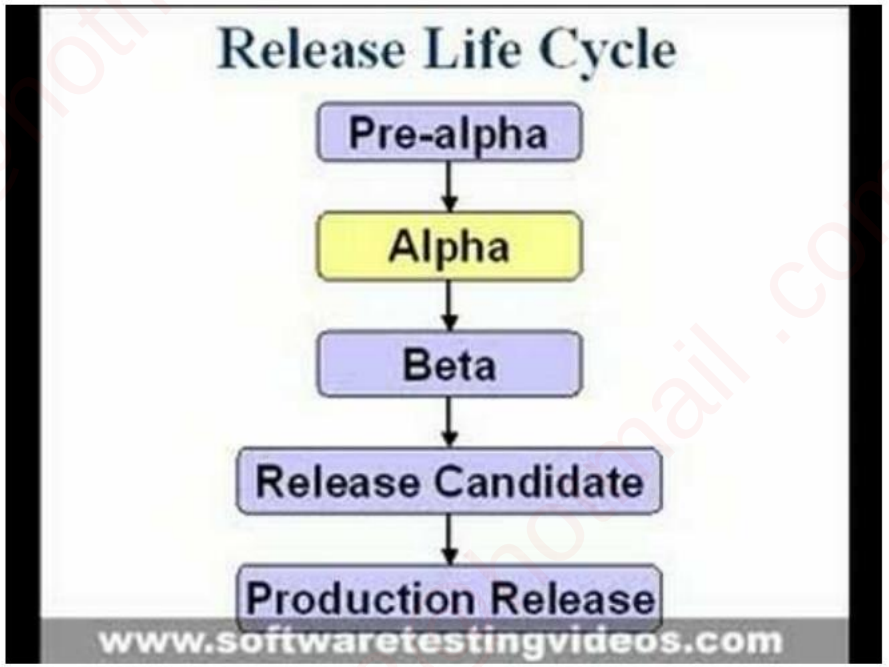
Ray Ma

# 今天我将学到

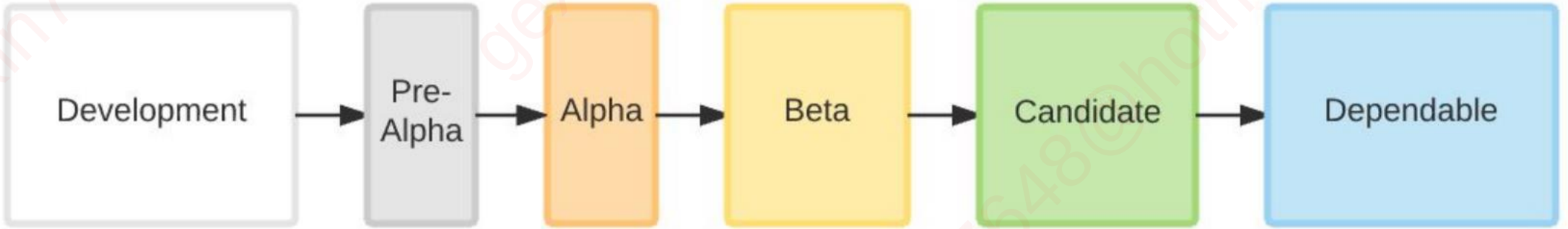
- 为什么要使用 Git?
- 如何设置 Git?
- Git 的基本使用方法
- 使用 Git branch 进行日常协作
- 使用 merge 和 rebase 合并分支
- 使用 GitHub 进行团队协作

# What is a Version Control System?



# Software release life cycle



# 为什么要使用 version control?



- 撤销改动或者回滚版本 (snapshot)
- 回溯历史 (A complete long-term history of every file that provides **traceability**)
- 协同合作（via branching strategies）
- 备份

# Version Control Glossary



- **Repository:** Your project.

  I created a new repository (repo) for my school project so we can collaborate more easily.

- **Diff:** The delta (additions and deletions) between two states of a project.

- **Commit:** A snapshot of your project's state at a point in history. Records the difference between two points in history with a diff.

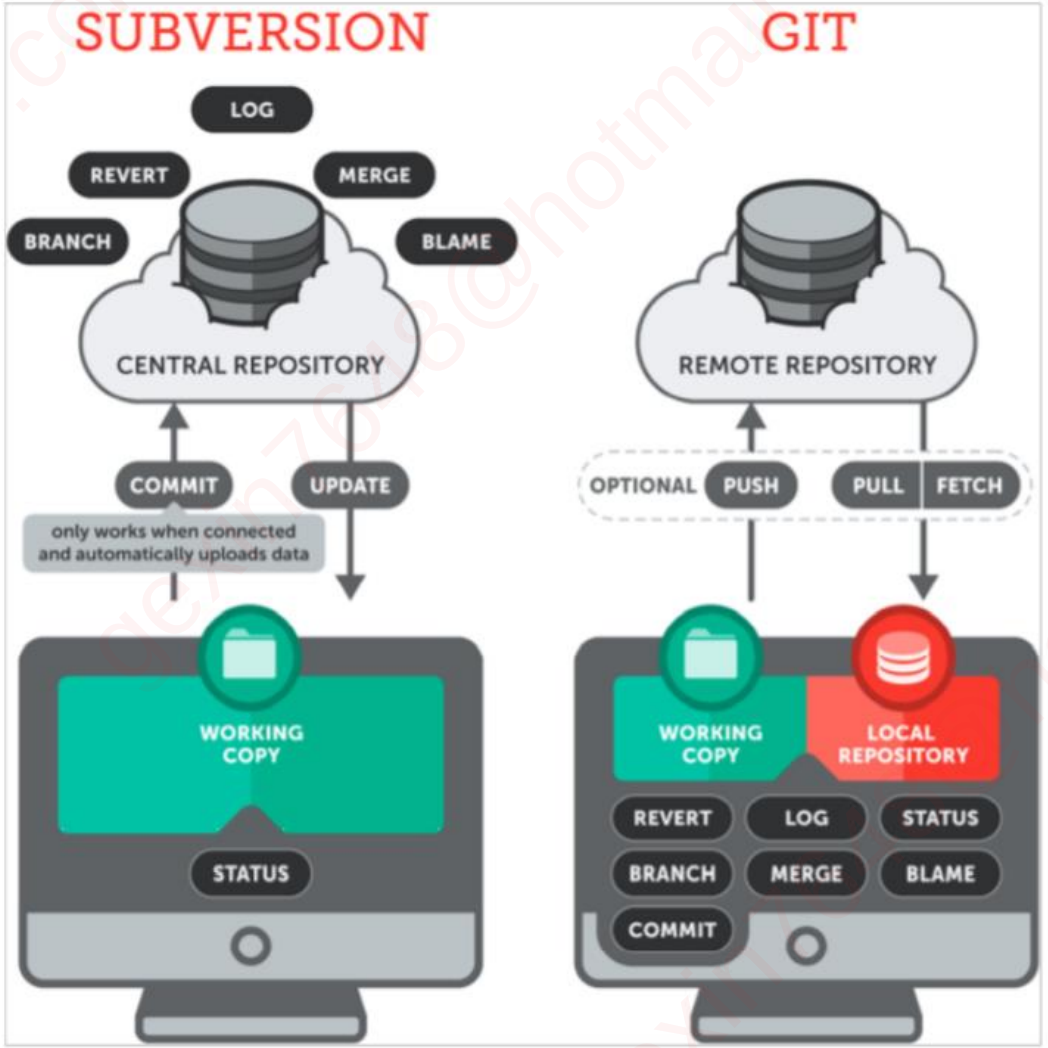
- **Branch:** Modifications to a project (main branch = trunk) made in parallel with the main branch, but not affecting the main branch.

# Version Control Glossary

- **Merge:** Introducing changes from one branch into another.
- **Clone:** Downloading a local copy of a project.
- **Fork:** Your own version of somebody else's project where you take the original code-base and make modifications. May include many changes or just a few bug-fixes. Sometimes you end up merging those changes back upstream.

# Centralized & Distributed





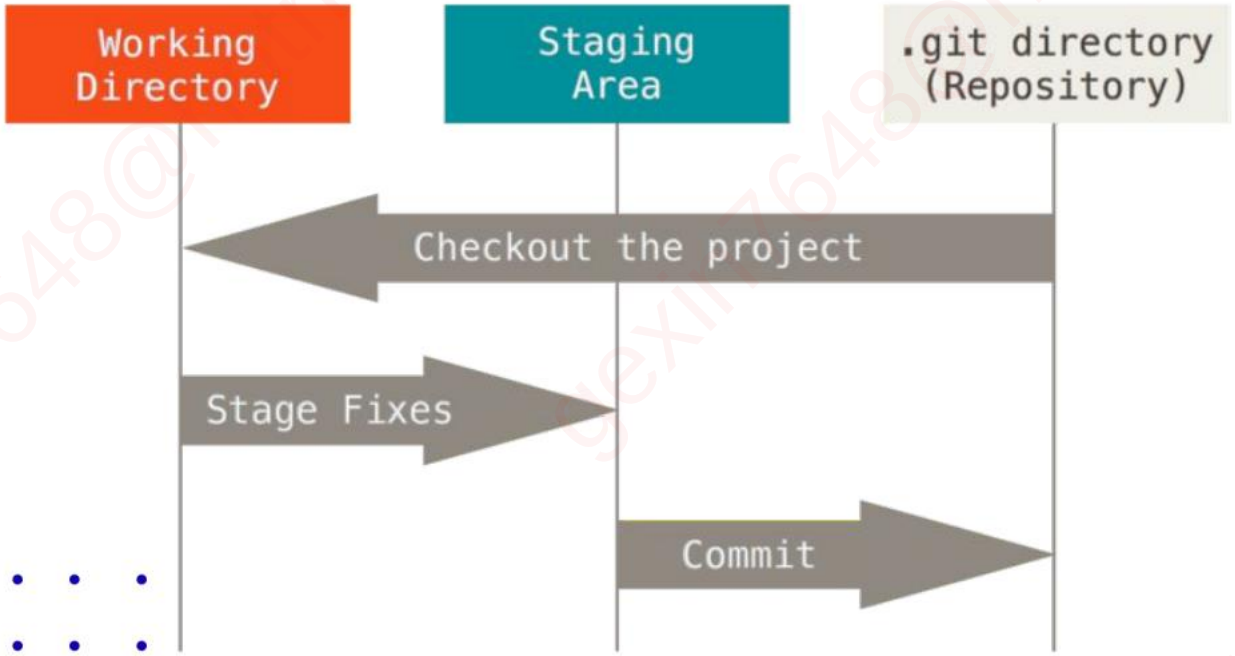
# What is Git?

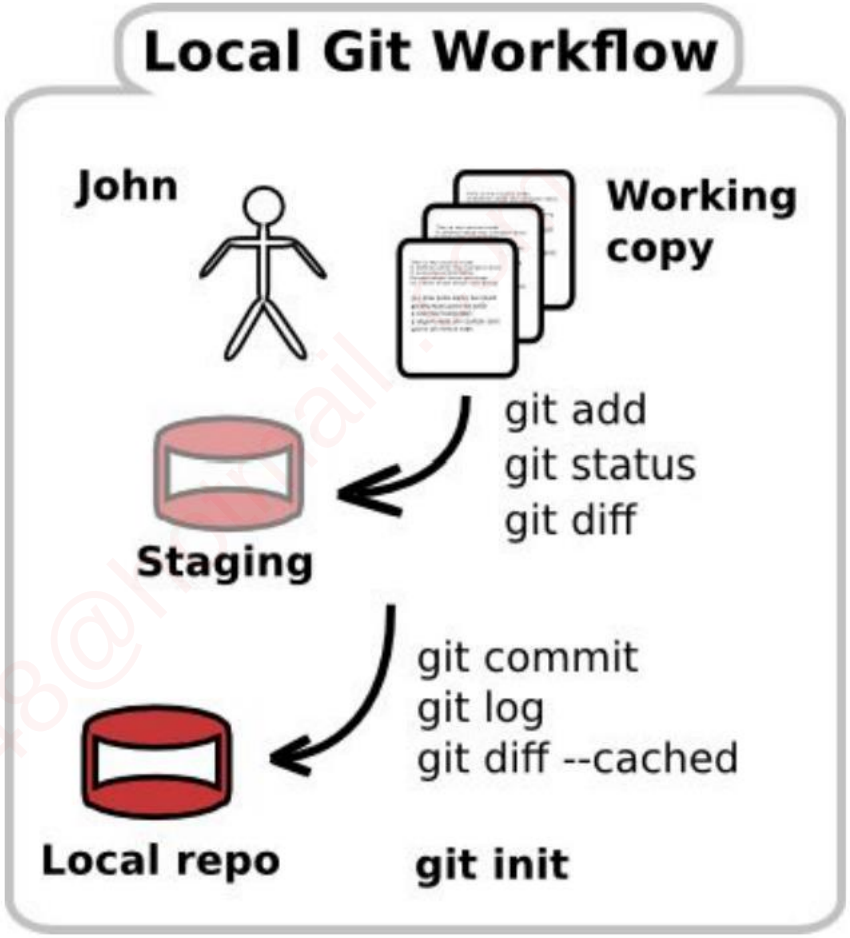
Git is a distributed version-control system for tracking changes in source code during software development.

It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

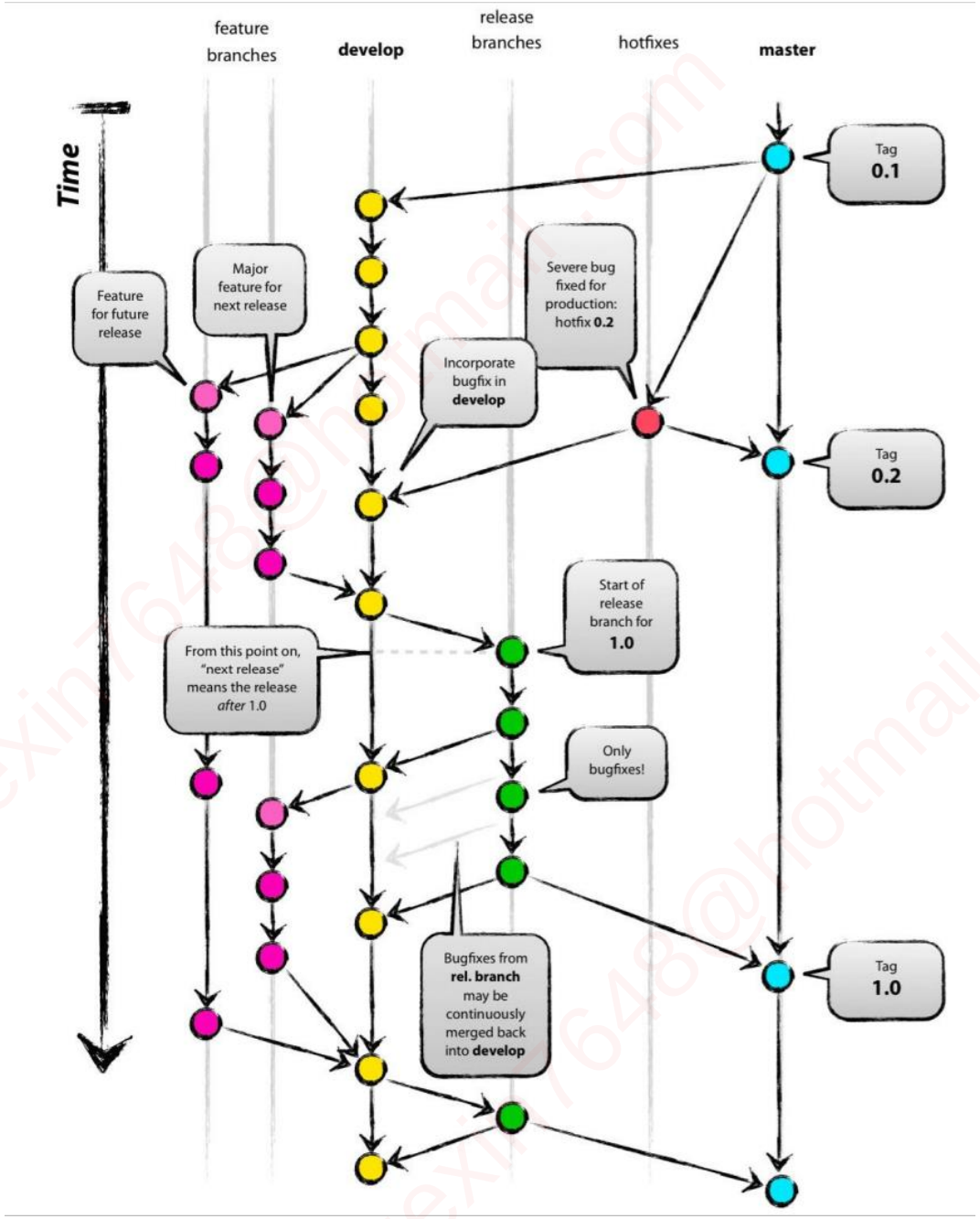
# Git workflow





# Git branch



# Git SCM Local

https://git-scm.com/downloads/guis

Cheat sheet

Full Reference

# Git and GitHub

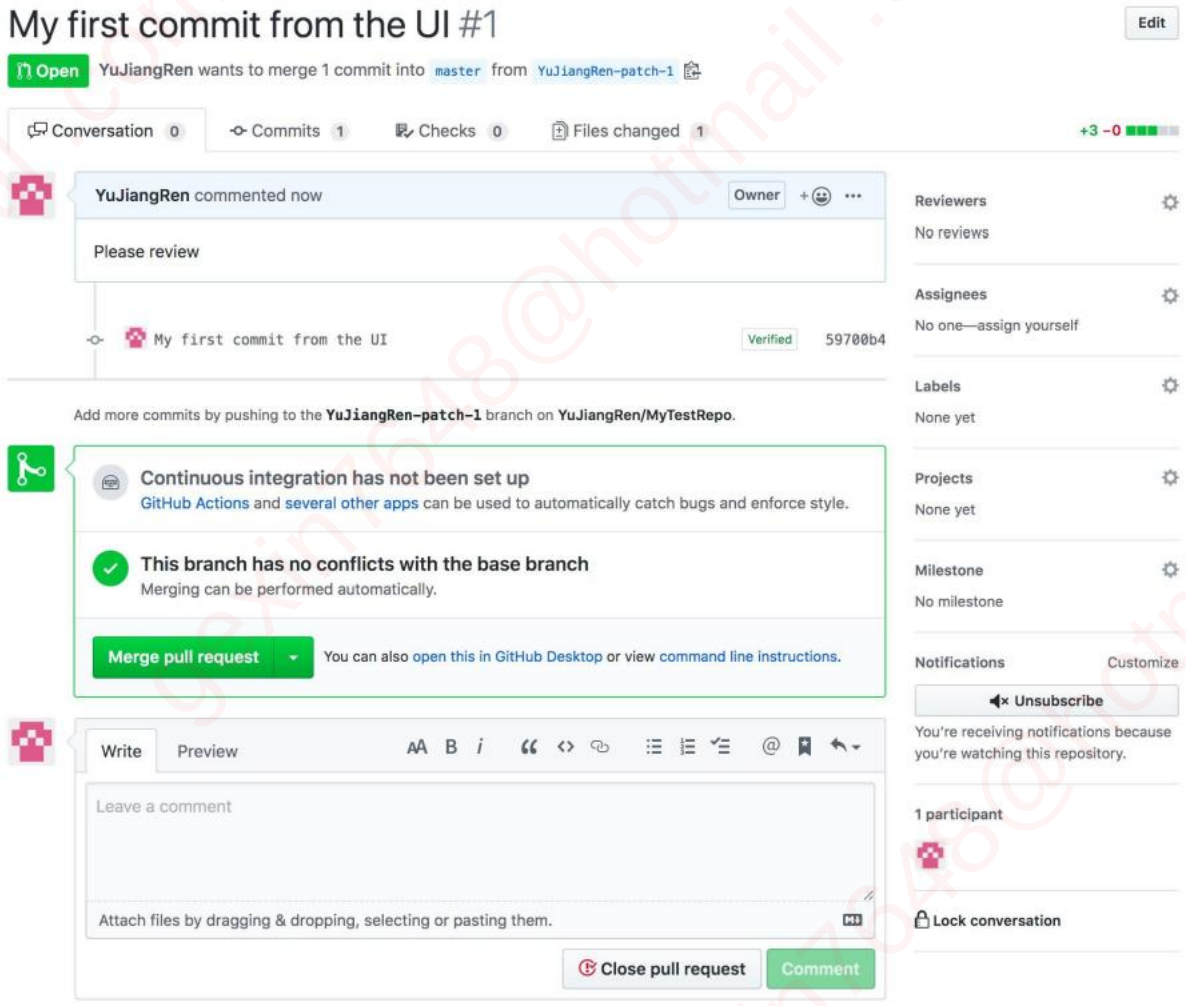

# Git and GitHub



# Git and GitHub



# GitHub PR



# Step 0: 一个点的历史 – Git Setup

# Git Global Setup

```
git config --global user.name "<Your-Full-Name>"

git config --global user.email "<your-email-address>"

git config --global color.ui auto

git config --global merge.conflictstyle diff3

git config --global core.editor “code --wait”
```

# Git - Repository

- A repository, or Git project, encompasses the entire collection of files and folders associated with a project, along with each file's revision history. The file history appears as snapshots in time called **commits**, and the commits exist as a linked-list relationship, and can be organized into multiple lines of development called branches.

- Because Git is a DVCS, repositories are self-contained units and anyone who owns a copy of the repository can access the entire codebase and its history.

- Using the command line or other ease-of-use interfaces, a git repository also allows for: interaction with the history, cloning, creating branches, committing, merging, comparing changes across versions of code, and more.

# Git local – hands on

- **git init** initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

```
$ mkdir learngit
$ cd learngit
$ pwd
/Users/jonny/learngit
$ git init
```

# Setting up a repository



```
git init
```

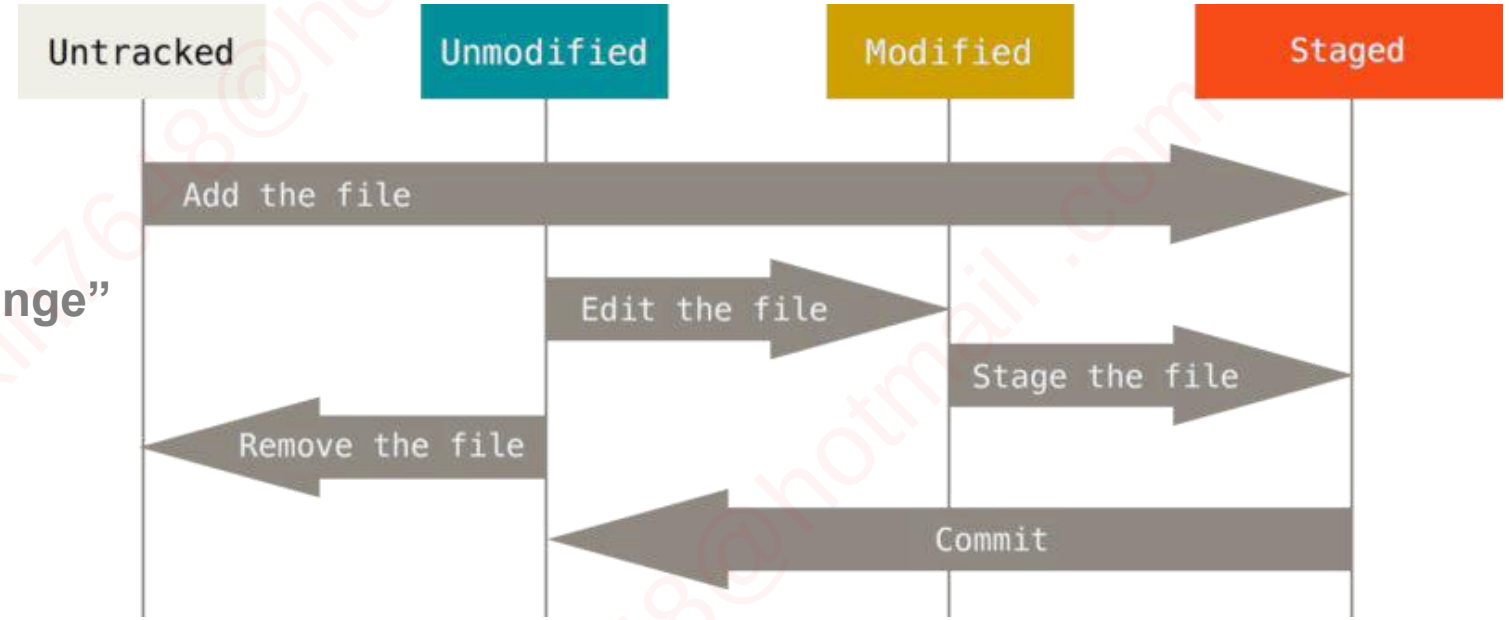
OR

```
git clone (remote repository only)
```

# Git local – hands on

- **Let's change README.md file and practice commands again.**

```
$ vi README.md
$ git status
$ git add . –p
$ git status
$ git commit –m "Another change"
$ git status
```

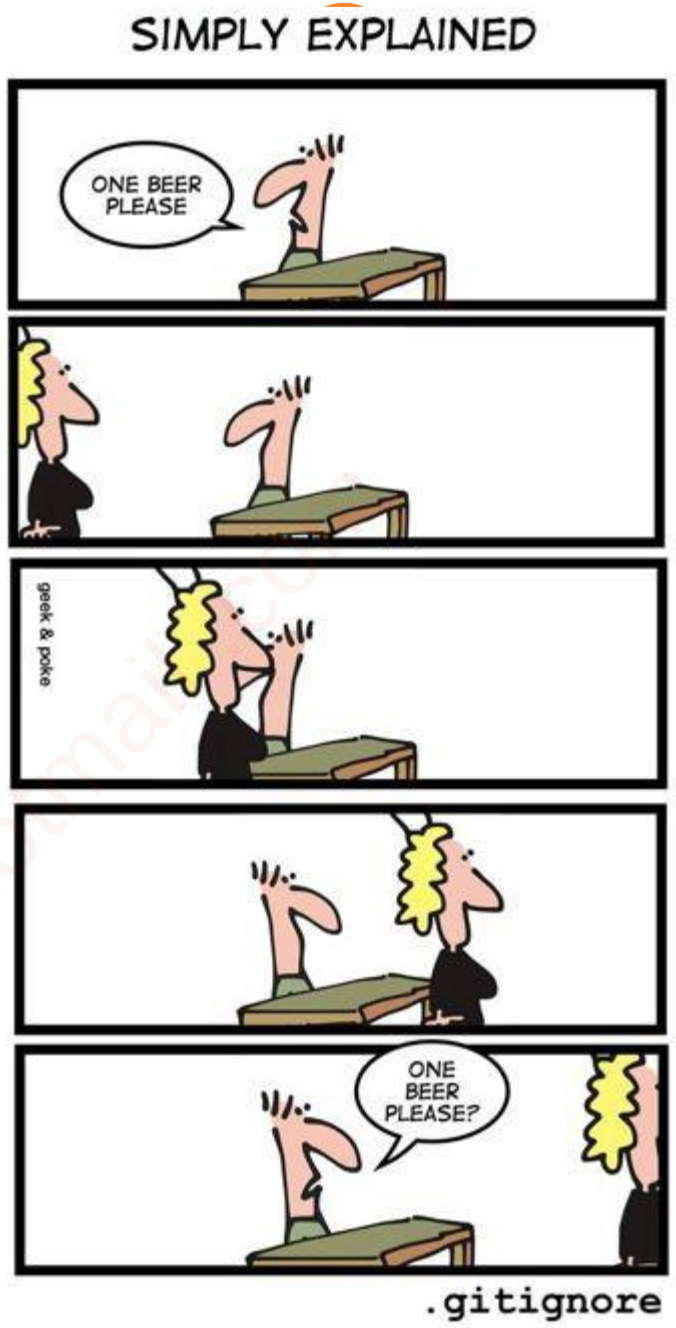


# Git local – hands on

## Ignoring Files

- Often, you'll have a class of files that you don't want Git to automatically add or even show you as being untracked. These are generally automatically generated files such as log files or files produced by your build system. In such cases, you can create a file listing patterns to match them named **.gitignore**

- **A collection of .gitignore templates:**
- https://github.com/github/gitignore

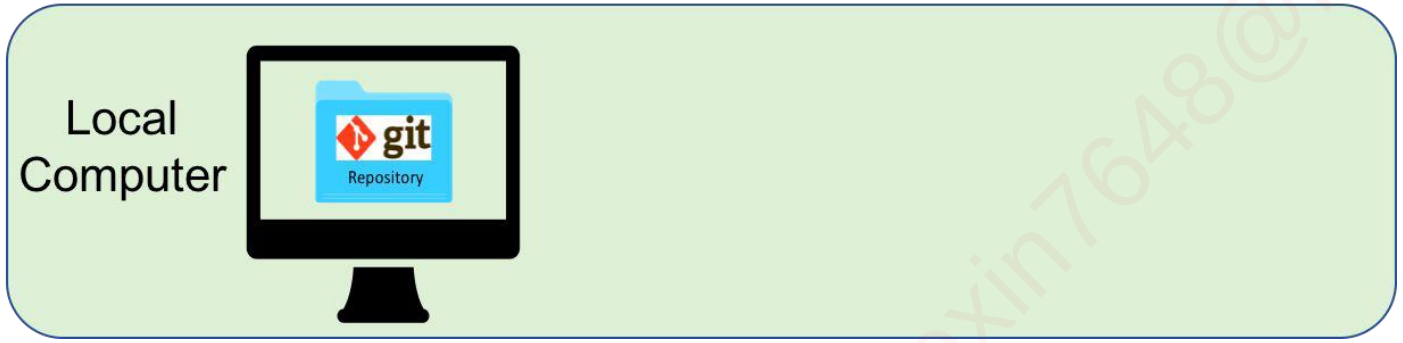


# Git local – hands on

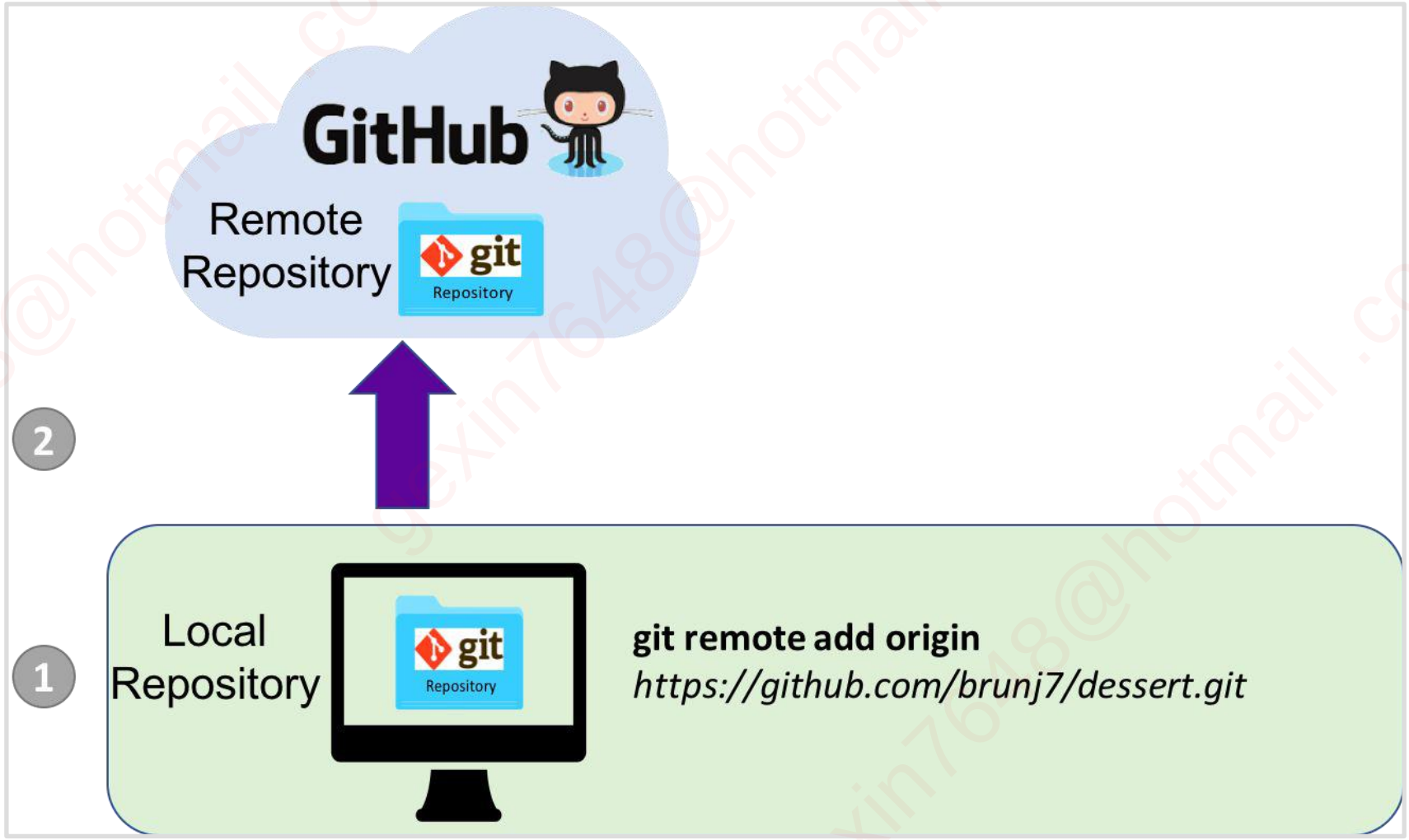
## Undoing things

- Undoing the last commit/comment: **git commit --amend**
- **git rm test.txt**
- **git clean** Remove untracked files from the working tree
- **git stash** Stash the changes in a dirty working directory away
- git stash
- git stash list
- git stash apply
- git stash pop  stash@{0}

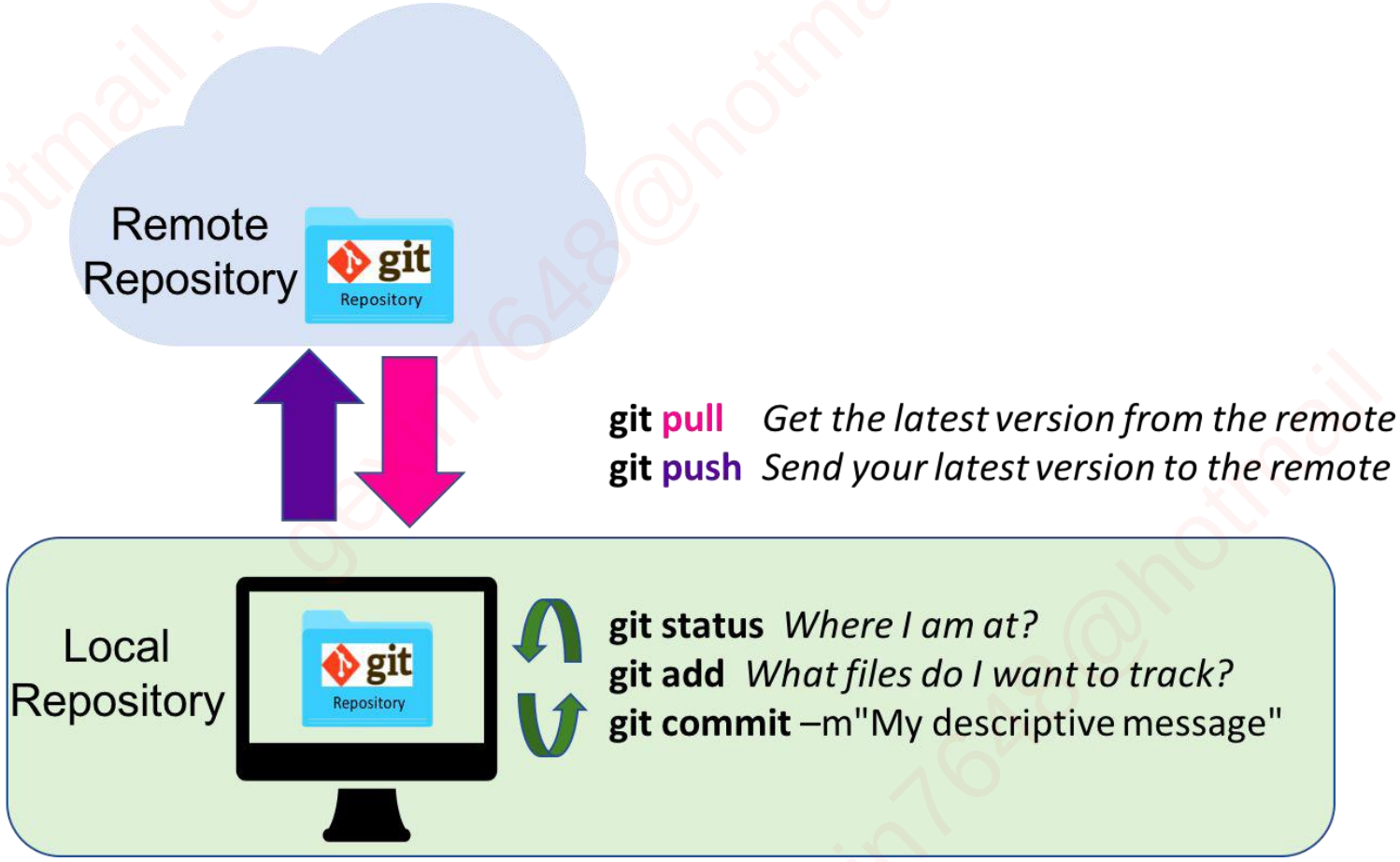
# Git Cloud Repository

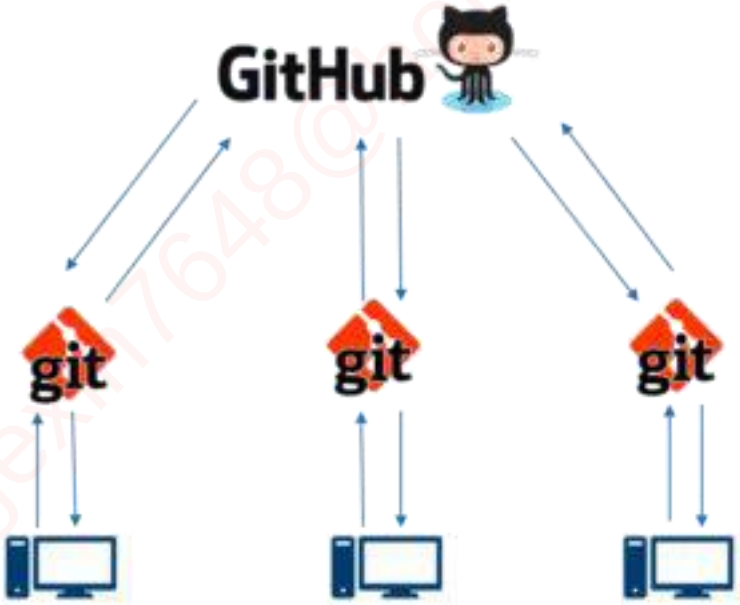
# Git Cloud Repository



# Git Cloud Repository



# Git Cloud Repository

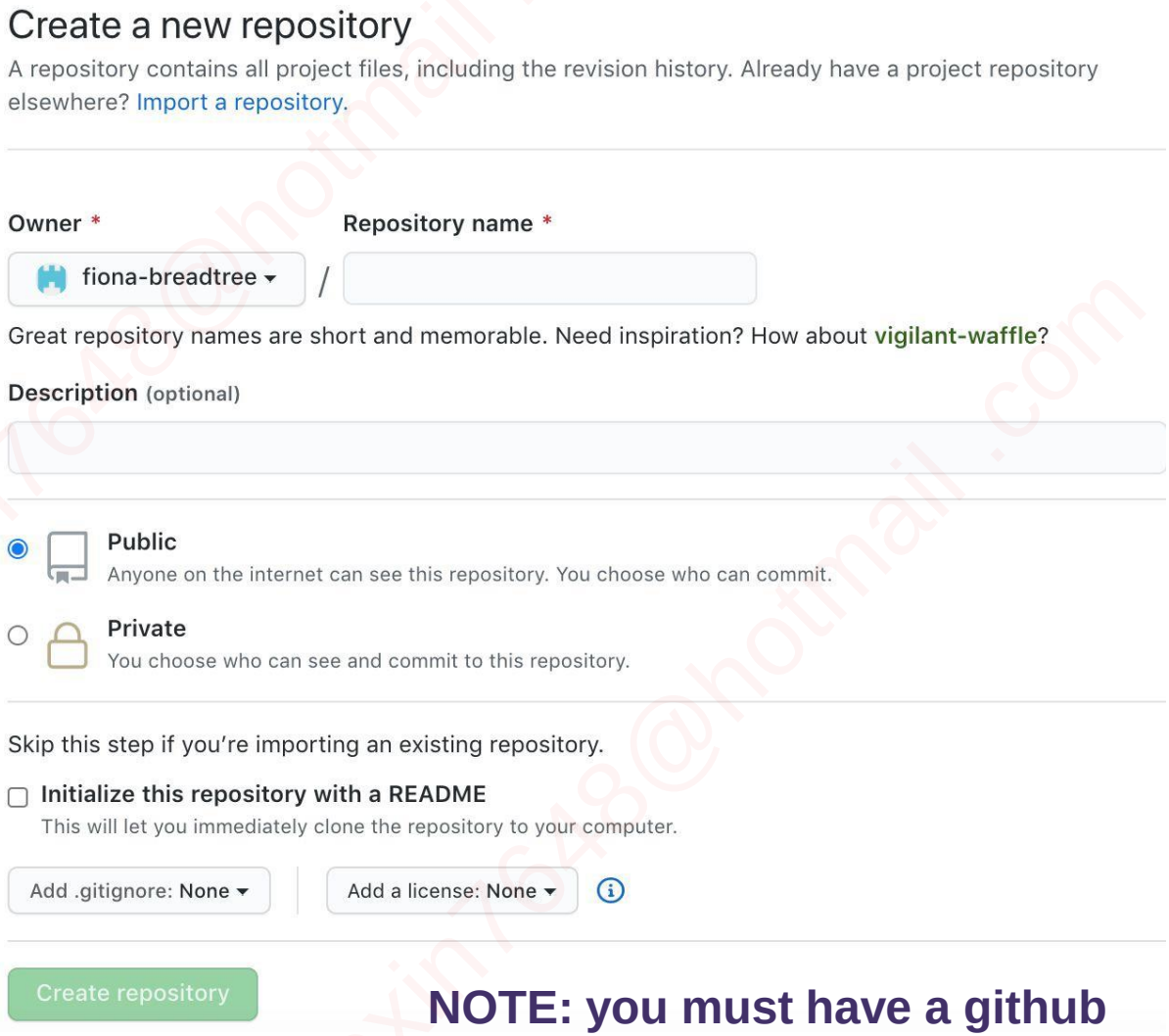


# Git remote – hands on

匠人学院 jiangren.com.au

## Create a "repository'

https://github.com/new




**NOTE: you must have a github account**

# Git remote – hands on

匠人学院
jiangren.com.au

**Copy (or clone) the repository to your local machine**

**- If you don't have it locally**

```
$ git clone https://github.com/username/reponame
```

**How to create personal access token:**

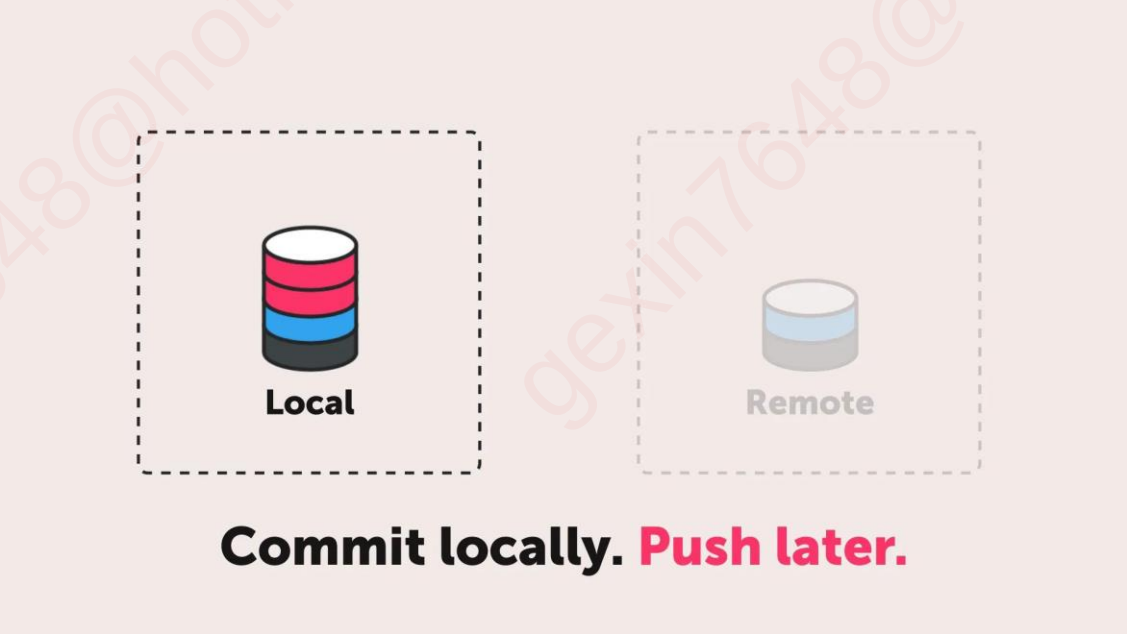
https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/creating-a-personal-access-token

**Or use SSH:**

https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh

# Git remote – hands on

- Add a file to your local repo and "commit" (save) the changes



# Git remote – hands on

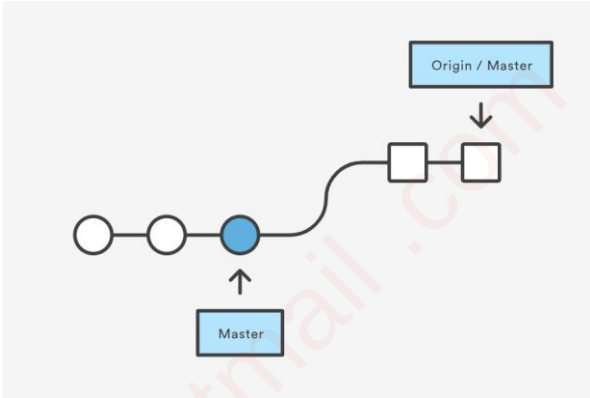
- **"Push" your changes to your master branch**

**git push** updates the remote repository with any commits made locally to a branch.

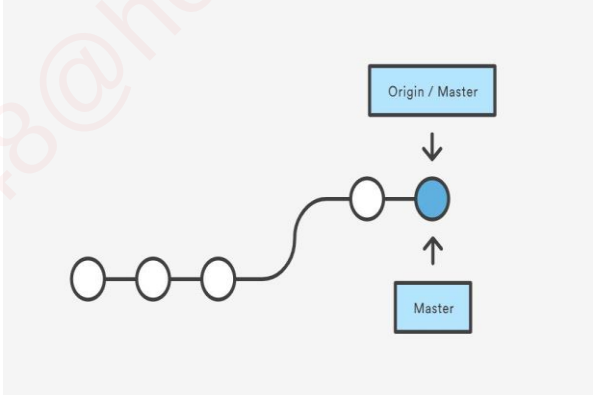
**git push origin main**

# Git remote – hands on

- **Make a change to your file with a git hosting tool (github UI) and commit**
- **"Pull" the changes to your local machine**




**git pull** updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

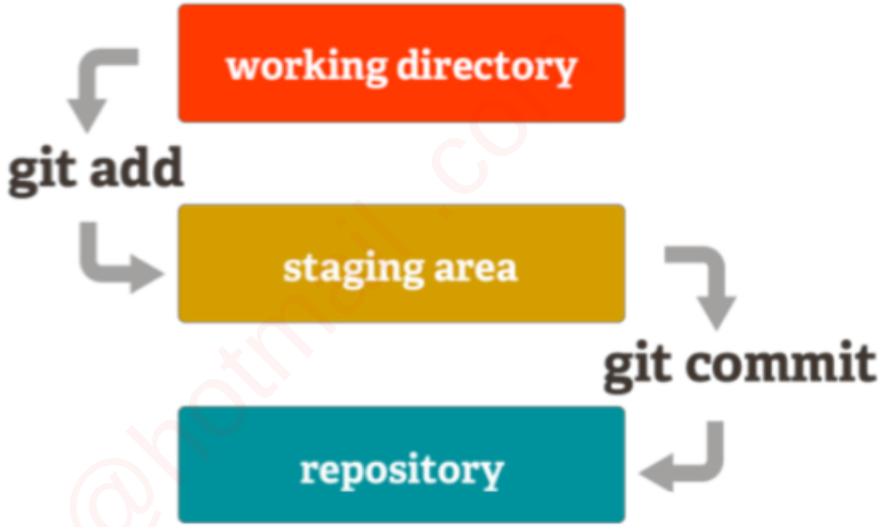



**git pull -r**

# Step 1: 一条线的历史 – commit

# Git工作原理：两步 commit



# Commit changes



```
git add

git commit

git rm (optional)

git stash (important!)
```

**to remove a file from a Git repository**

git stash list
git stash pop
git stash apply --index

# Commit message 要有意义

# Git undo changes

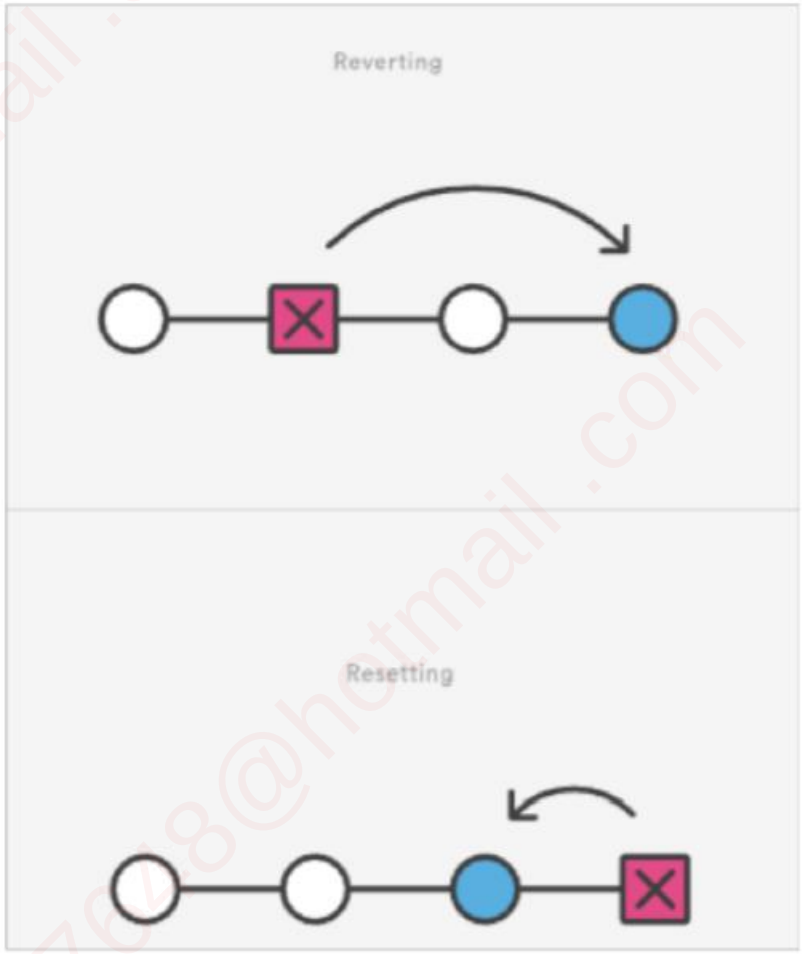


```
git checkout

git clean

git revert
```

用一个新的commit对历史记录进行回滚

```
git reset
```

从历史记录中删除commit



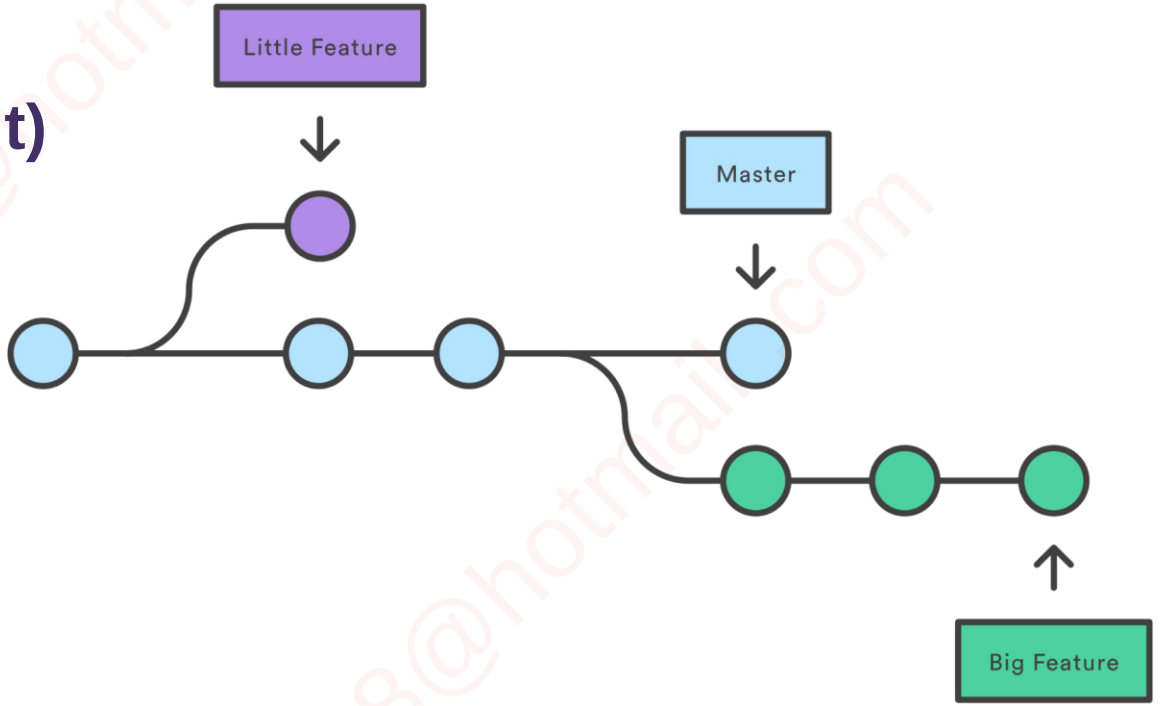
# Git status and history

```
git status

git diff

git log
```

# Step 2: 两条线的历史 – branch & merge

# Git with branching

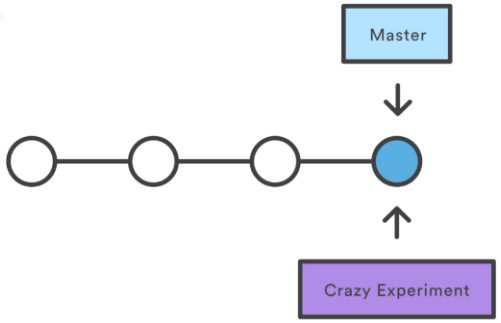
- Create a branch (git branch)
- Checkout a branch (git checkout)
- Merge a branch (git merge)
- Rebase a branch (git rebase)



# Git with branching

- **Create a branch (git branch)**
- **Checkout a branch (git checkout)**

**git branch**
**crazy-experiment**




Note that this only *creates* the new branch. To start adding commits to it, you need to select it with **git checkout**, and then use the standard git add and git commit commands.

**git checkout –b**
**crazy-experiment**

# Branching

```
git branch

git branch -d

git checkout -b
```





https://www.conventionalcommits.org/en/v1.0.0-beta.4/

# Semantic branch names

<type>/<ticket-number>-<title>
e.g. **feat/JR-101-create-header-for-home-page**

- feature
- fix/bugfix/hotfix
- chore

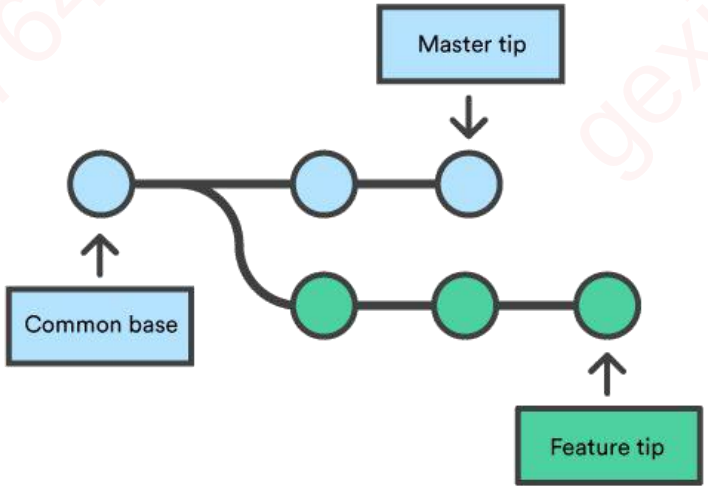


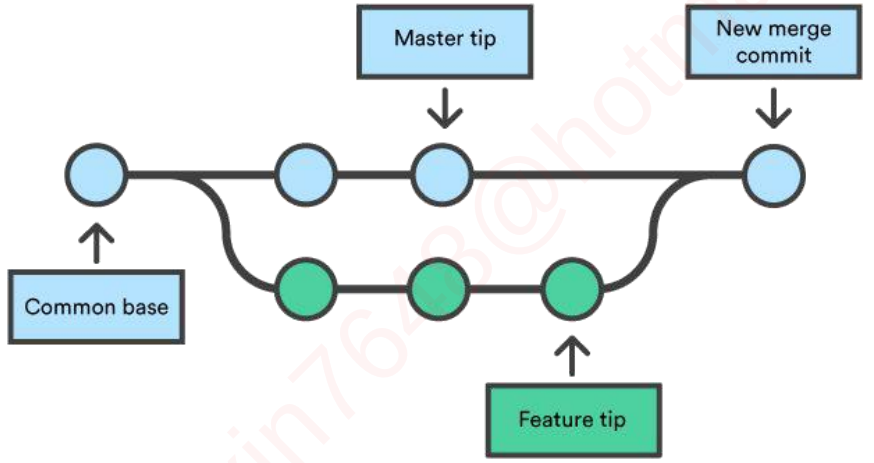
https://www.conventionalcommits.org/en/v1.0.0-beta.4/

# Git with branching

- **Merge a branch (git merge)**

Git merge lets you take the independent lines of development created by git branch and integrate them into a single branch. Git merge will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches.



# Git with branching

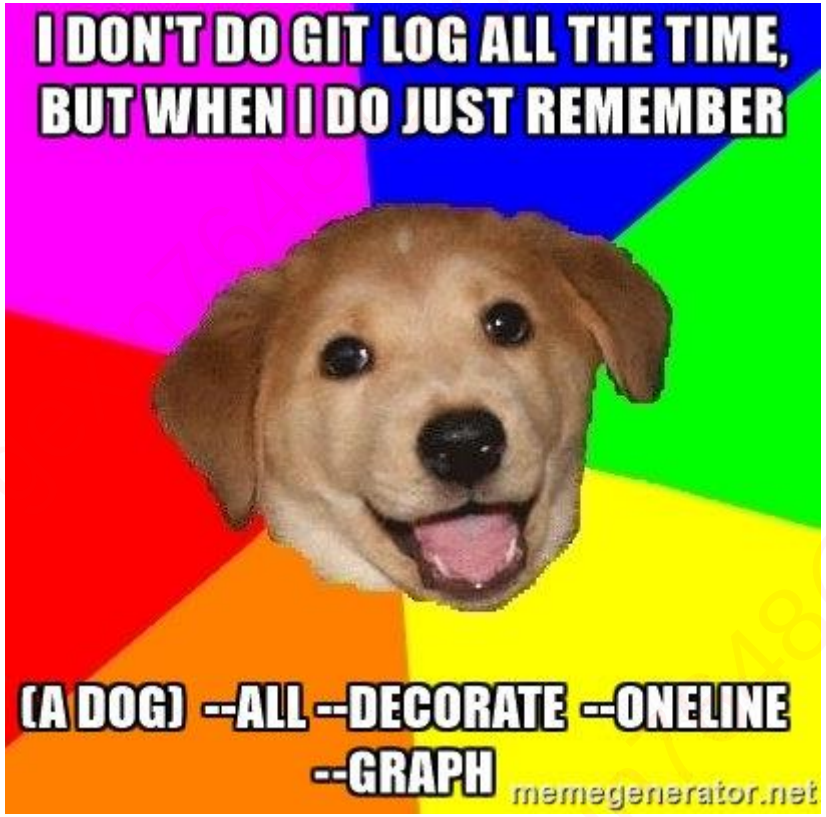
- Merge a branch (git merge)

```
# Start a new feature
git checkout -b new-feature
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Merge in the new-feature branch
git checkout main
git merge new-feature
git branch -d new-feature
```

# Git Graph



```
git log --all --decorate --oneline --graph
```



# Merge conflicts!!!




如果不想解决冲突:

git merge <branch name> --abort

git merge <branch name> --overwrite-ignore

git merge <branch name> --no-overwrite-ignore

# Git with branching - Conflicts



- **Merge a branch (git merge)**

**BUT~~~ things cannot always be so easy ☹**

**What if the same file has been modified by other people?**

```
git checkout -b new-feature
# Edit some files
git commit -m "Finish a feature"
# Develop the main branch
git checkout main
# Edit the same file
git add <file>
git commit -m "Make some super-stable changes to master"
# Merge in the new-feature branch
git merge new-feature
```

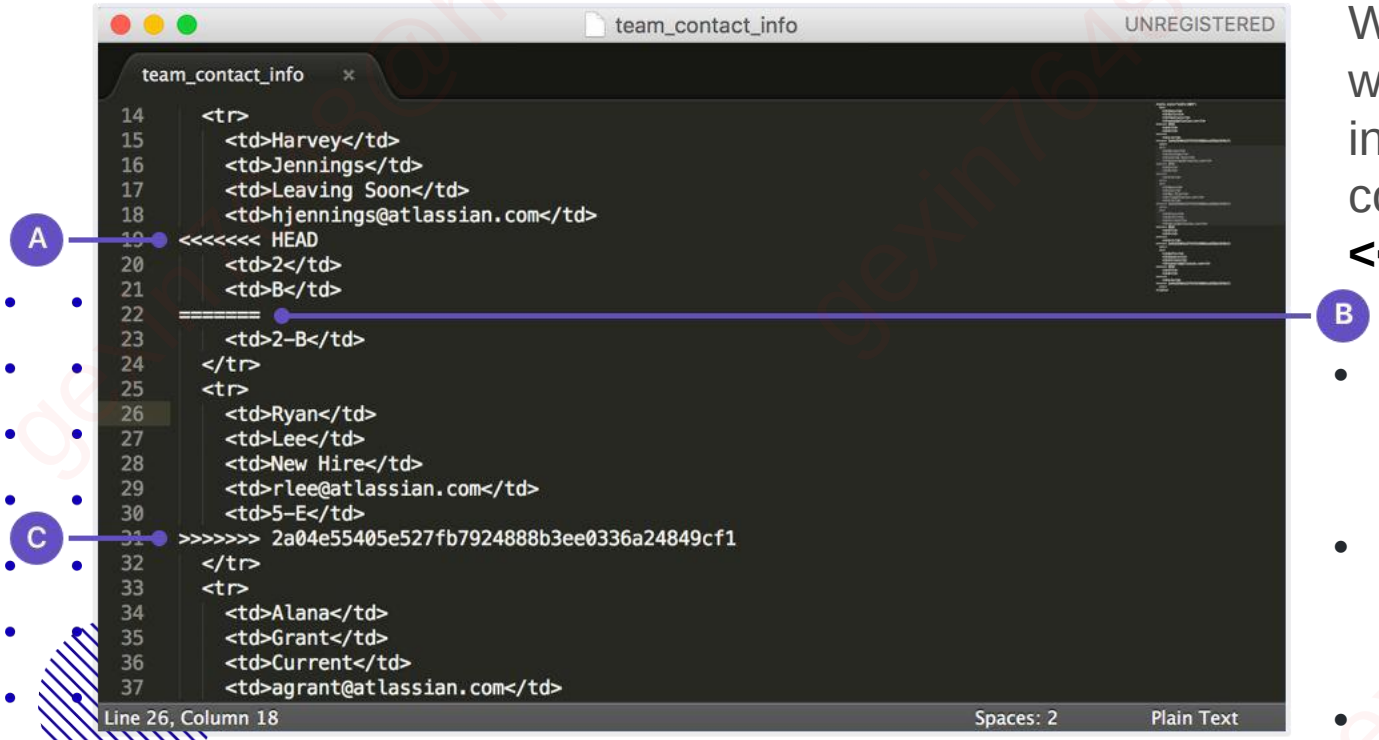
When you encounter a merge conflict, running the git status command shows you which files need to be resolved.

# Git with branching – Solve Conflicts



- **Merge a branch (git merge)**

When you encounter a merge conflict, running the **git status** command shows you which files need to be resolved.



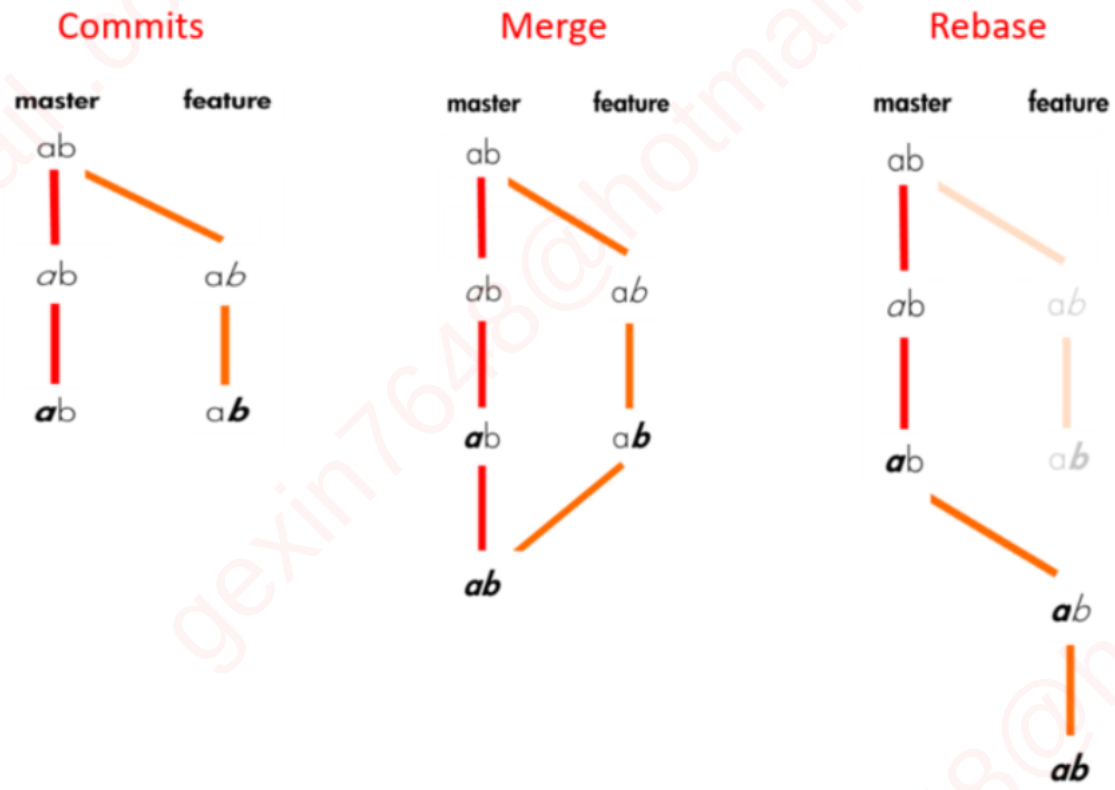

When Git encounters a conflict during a merge, It will edit the content of the affected files with visual indicators that mark both sides of the conflicted content. These visual markers are: **<<<<<<<**, **=======**, and **>>>>>>>**

- **A.** The beginning of the change in the HEAD branch. In this case, HEAD represents the active branch into which you're merging.
- **B.** The end of the change in the active branch and the beginning of the change in the non-active branch.
- **C.** The end of the change in the non-active branch.

# 两种workflow: merge 和 rebase

# Merging vs. Rebasing





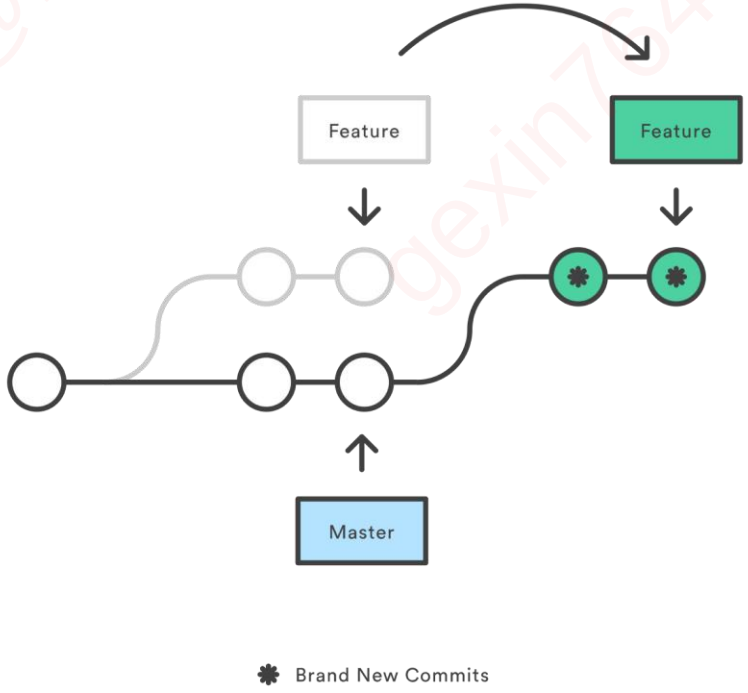

Preserve the complete history of your project.

A clean, linear history free of unnecessary merge commits.

# Git with branching

- **Rebase a branch (git rebase)**

**Git rebase** Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow.




git checkout feature
git rebase main
git log

**Merge** is always a forward moving change record. Alternatively, **rebase** has powerful history rewriting features.

# Git with branching

匠人学院
jiangren.com.au

- **Rebase a branch - git rebase interactive mode**

  Interactive rebasing gives you the opportunity to alter commits as they are moved to the new branch. This is even more powerful than an automated rebase, since it offers complete control over the branch's commit history. Typically, this is used to clean up a messy history before merging a feature branch into master.

```
git checkout feature
git rebase -i main

git rebase –i HEAD~1
```

**The golden rule of git rebase is to**
**never use it on *public* branches.**

# Step 3: 多条线的历史 – 远程协作

# Connect to remote repo



```
git clone

git remote add <name> <url>

git remote rm <name>

git remote rename <old-name> <new-name>
```

# Set up tracking branch



```
git branch -u <name>/<branch>
```

也可以在push的时候建立这种联系

# Update local from remote

```
git fetch

git pull
```

# Update remote from local

```
git push <name> <branch>

git push <name> <local_branch>:<remote_branch>
```

# Delete a remote branch



```
git push -d <name> <branch>

git push <name> :<branch>
```

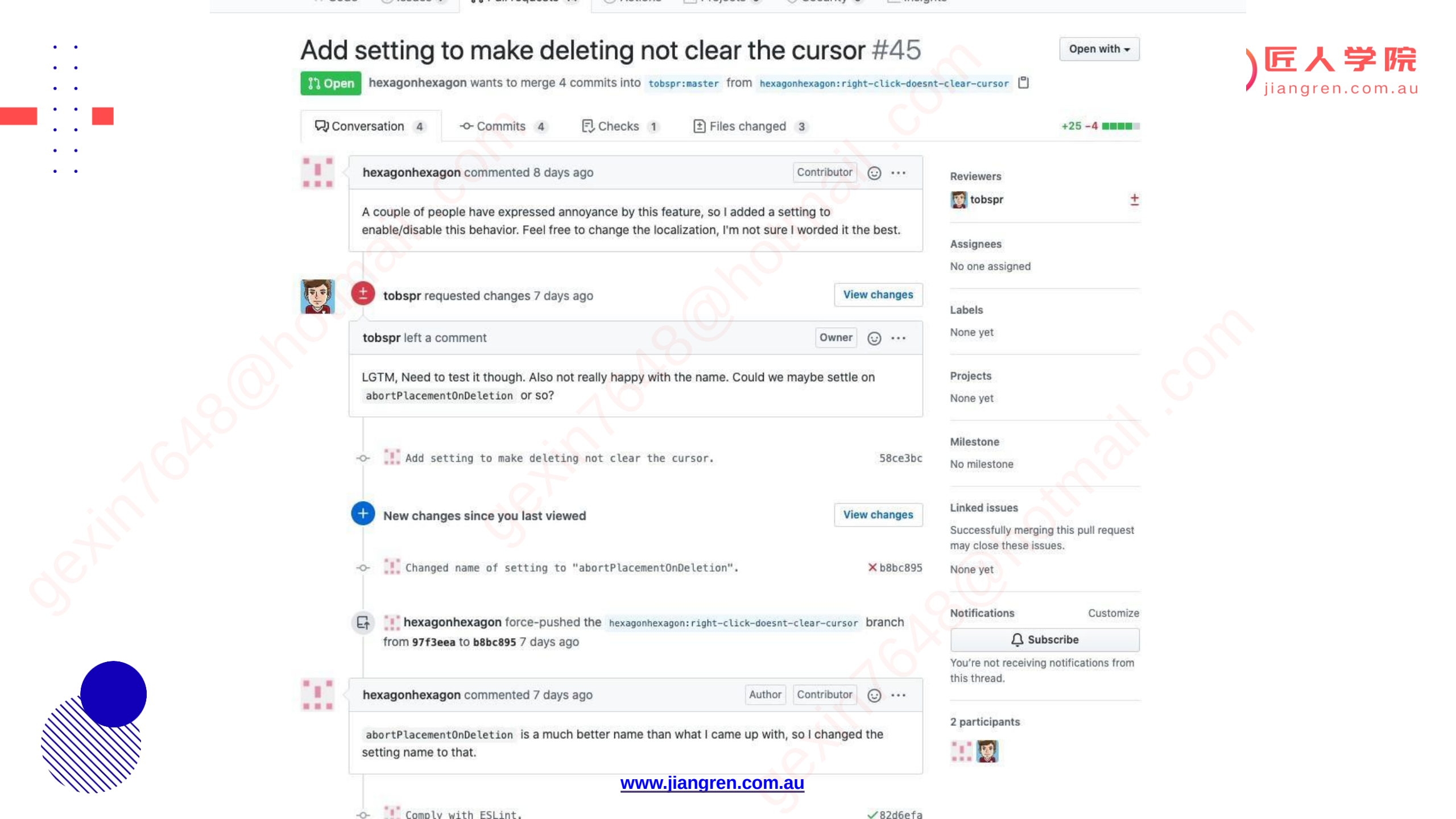
# Pull Request (PR)

Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

# Pull Request (PR)

**Benefits:**

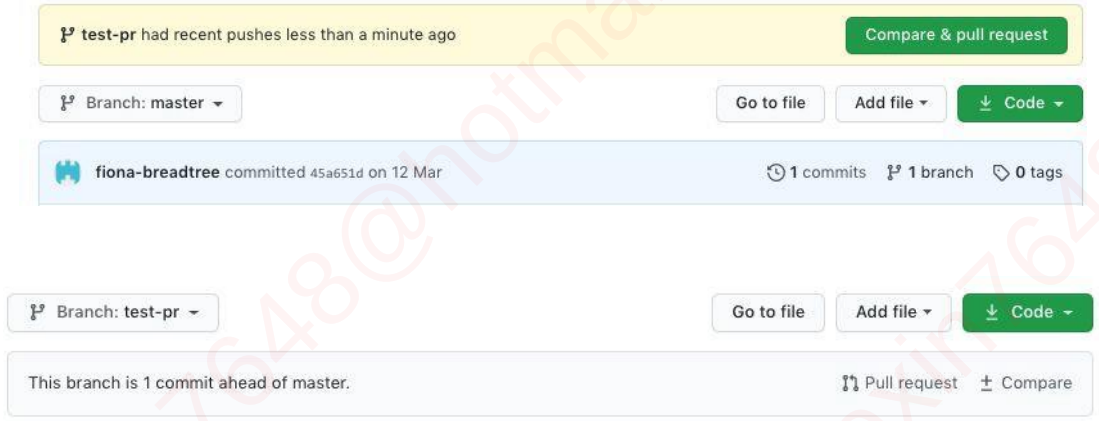
- Peer Review
- Sufficient testing and better stability
- Reducing conflicts Continuous Delivery
- Clearer responsibility

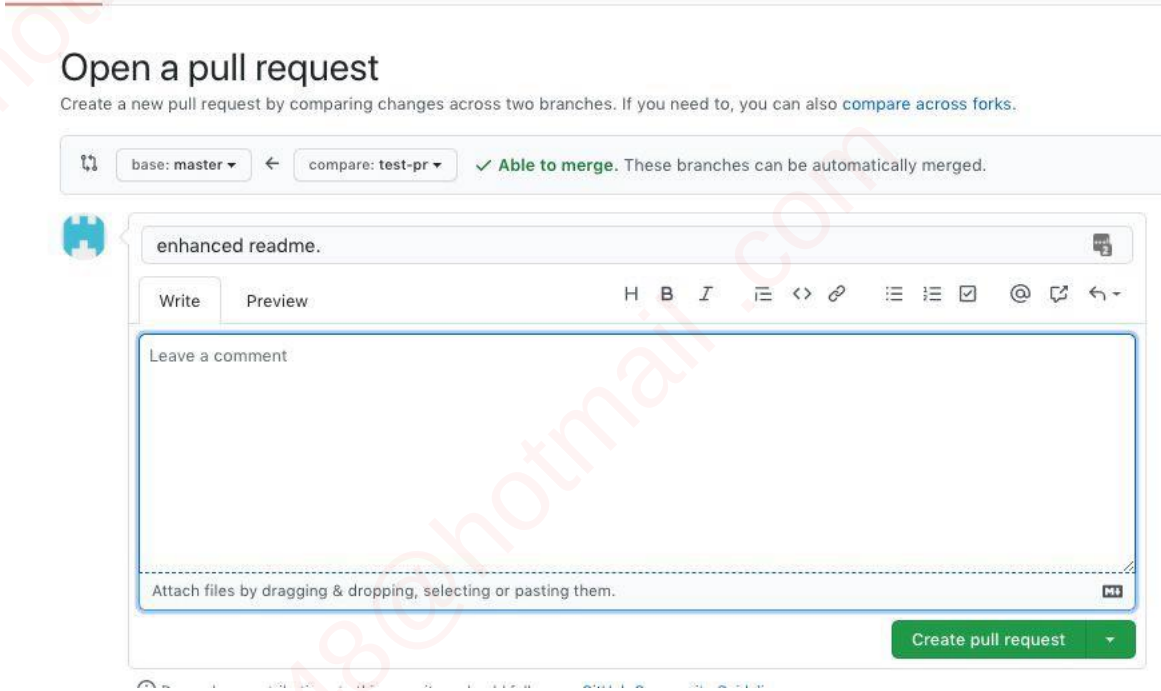


# Pull Request (PR)

匠人学院
jiangren.com.au

When you push your branch, got the github repo:



# Pull Request (PR)

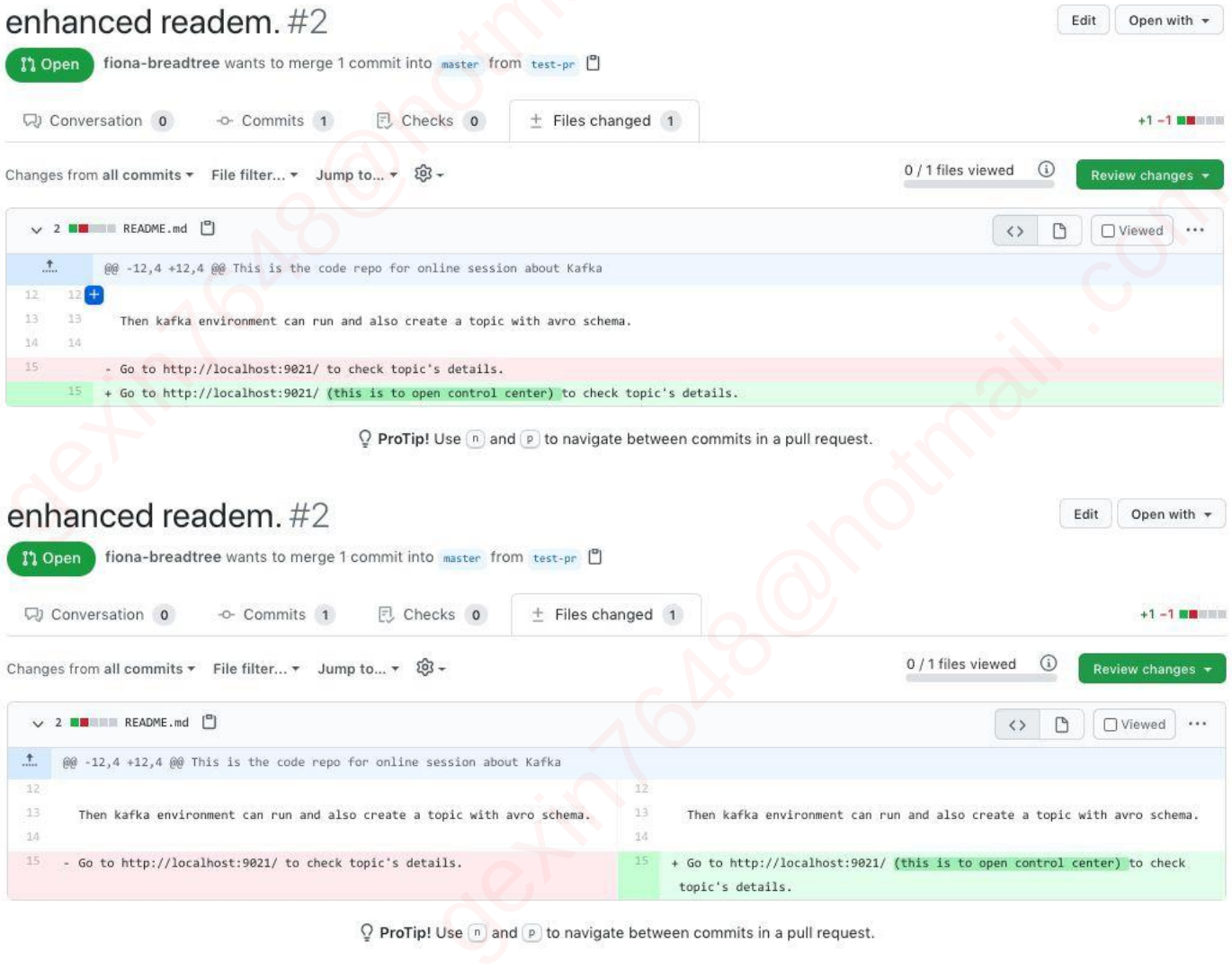
Before you create Pull Request, please check:

- Follow coding standard (code style, naming convention, etc. )
- Have tests
- No conflict
- **DO NOT** leave comment blank – add summary of this PR

# Pull Request (PR)

Code Review:

- Two ways to show diff
- Single comment vs. multiple

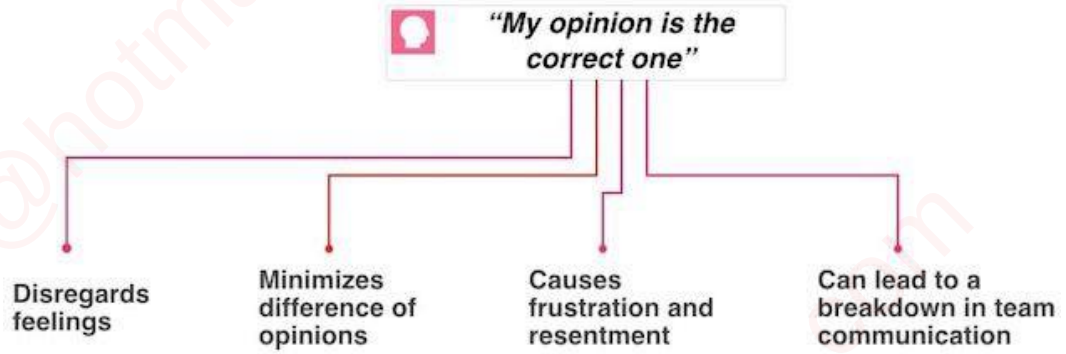


# Pull Request (PR)

匠人学院
jiangren.com.au

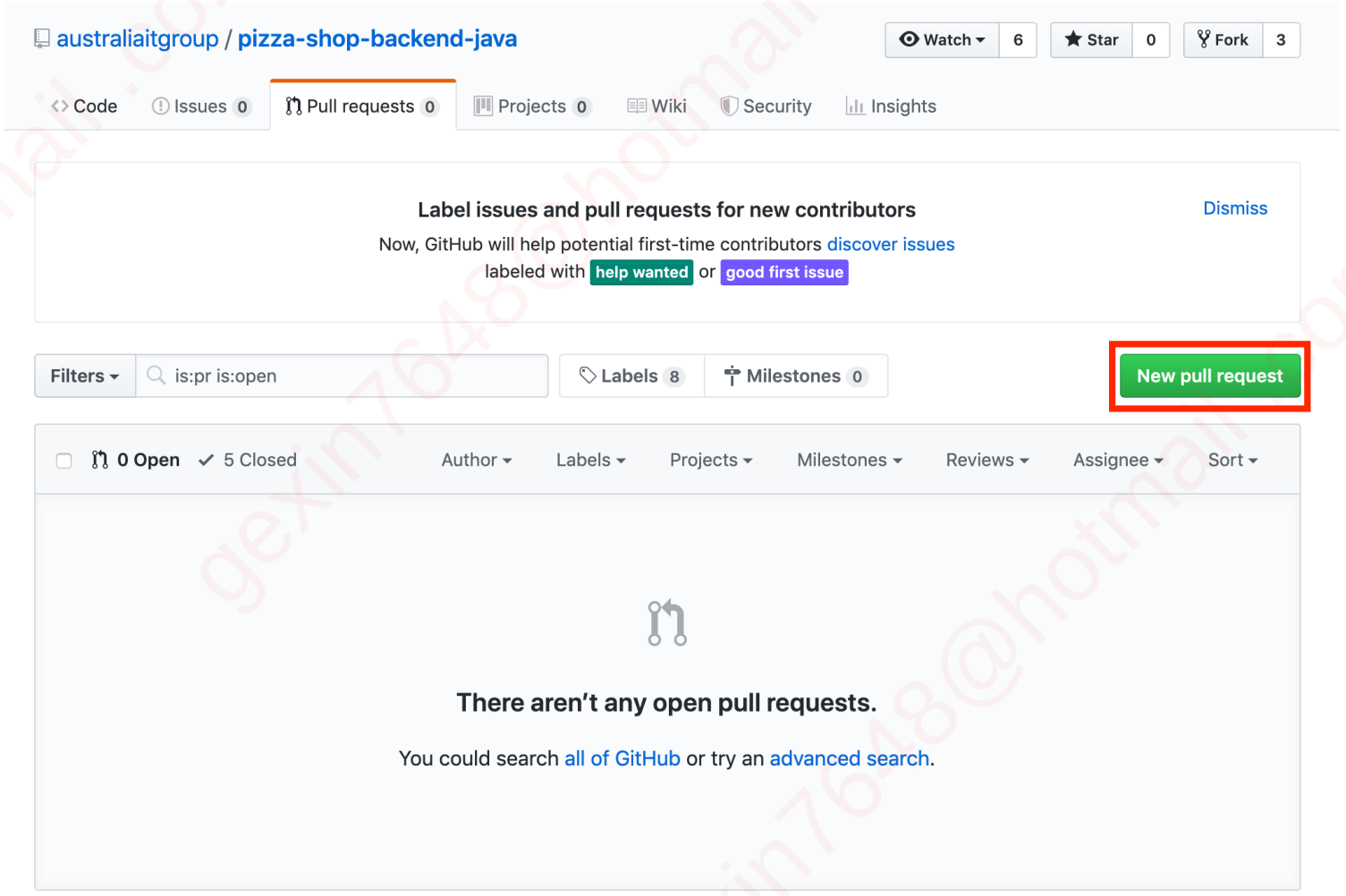
**Comments for Code Review:**

- **LGTM – look good to me**
- **:+1**
- **Markdown**




- The reviewer should identify errors that will cause an issue in production.
- The reviewer should verify that the stated goal of the code change aligns with the changes being made.
- The reviewer should verify that any changes align with the team's coding standards.
- The reviewer should look for anything they personally disagree with.

# Create a pull request (PR)



# Create a pull request (PR)



## Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base repository: MagicCube/tensorflow-rex-r... | base: gh-pages

head repository: australiaitgroup/tensorflow-... | compare: master ✓ Able to merge. These branches can be automatically merged.

Create pull request  Discuss and review the changes in this comparison with others.

5 commits | 10 files changed | 0 commit comments | 1 contributor

Commits on Apr 15, 2018

| | | |
|---|---|---|
| MagicCube | Update generationEL | 5773f8c |
| MagicCube | Add fit() | ab95fb6 |
| MagicCube | Add ; | 5359664 |
| MagicCube | Change train() to fit() | 2b4f069 |

Commits on Apr 28, 2018

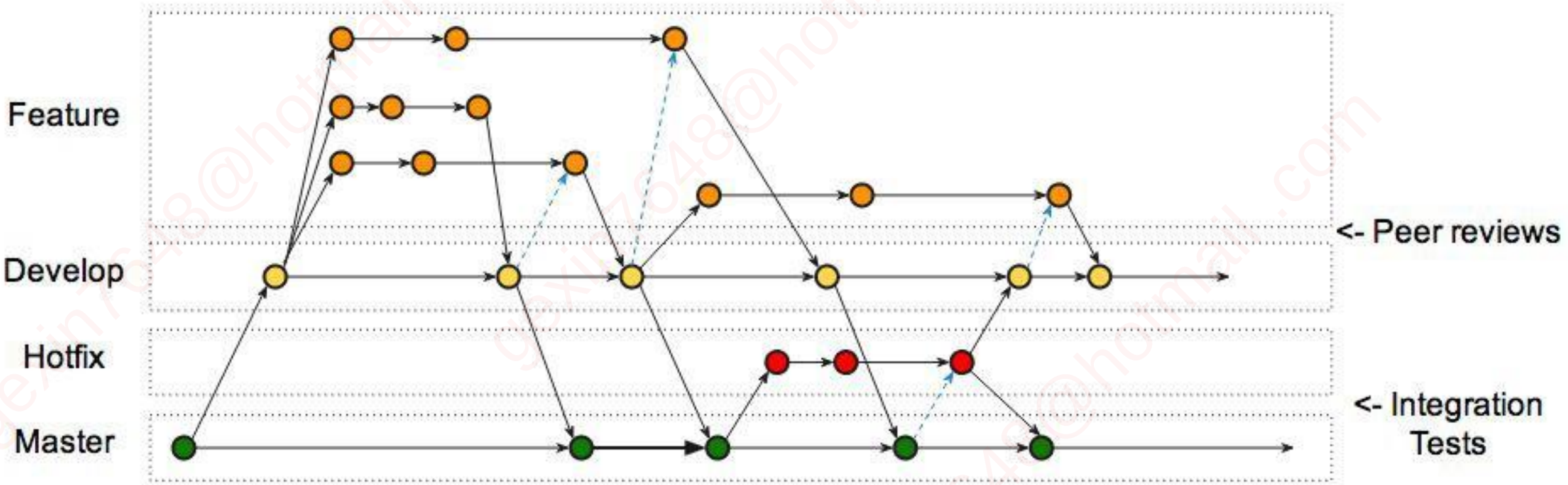
| | | | |
|---|---|---|---|
| MagicCube | Update README.md | Verified | 54b87fe |

Showing 10 changed files with 25 additions and 17 deletions.  Unified | Split

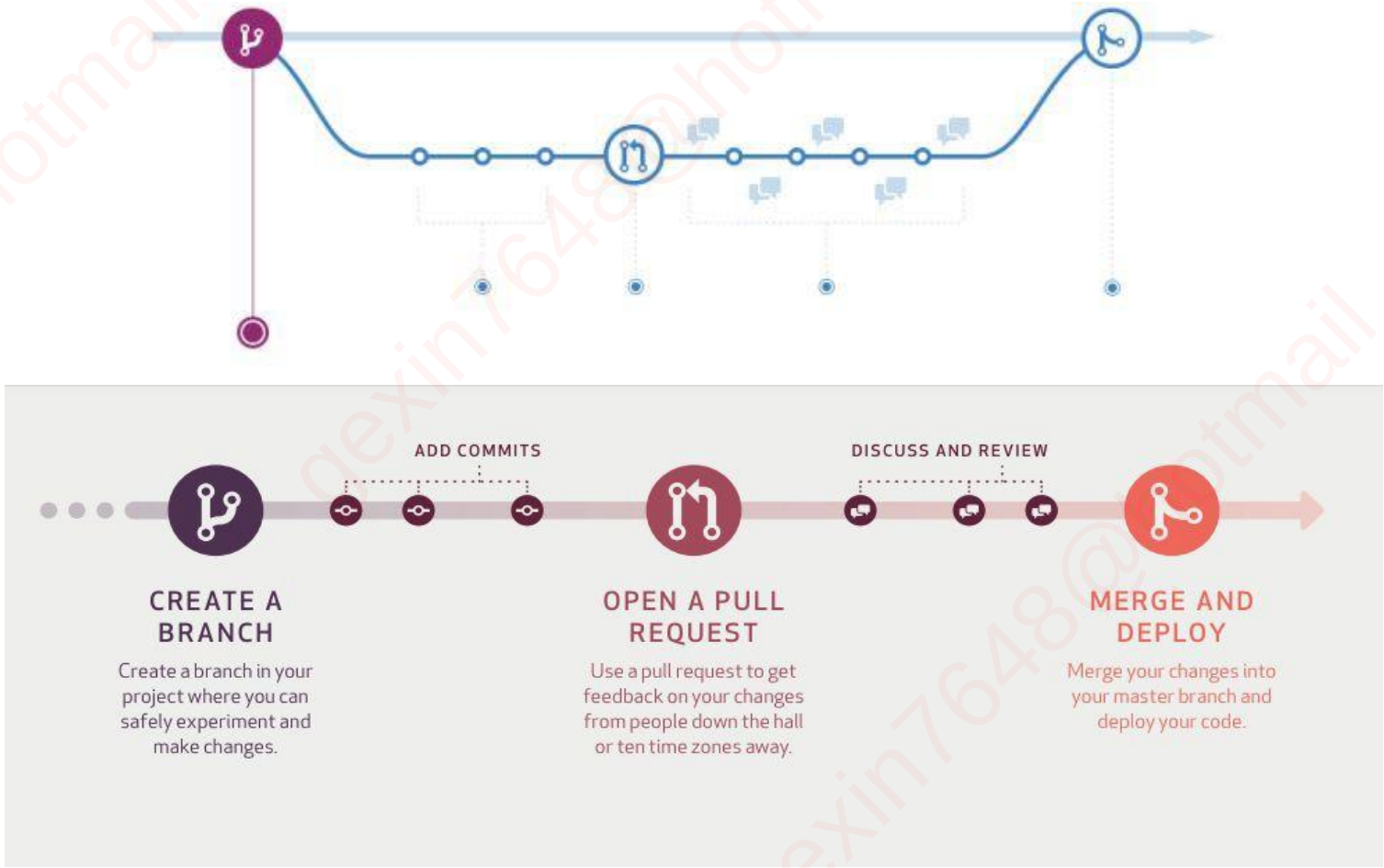
2 README.md

```
@@ -1,4 +1,4 @@
-# t-rex-run
+# tensorflow-rex-run

 A TensorFlow.js based AI player platform for T-Rex Runner.
```

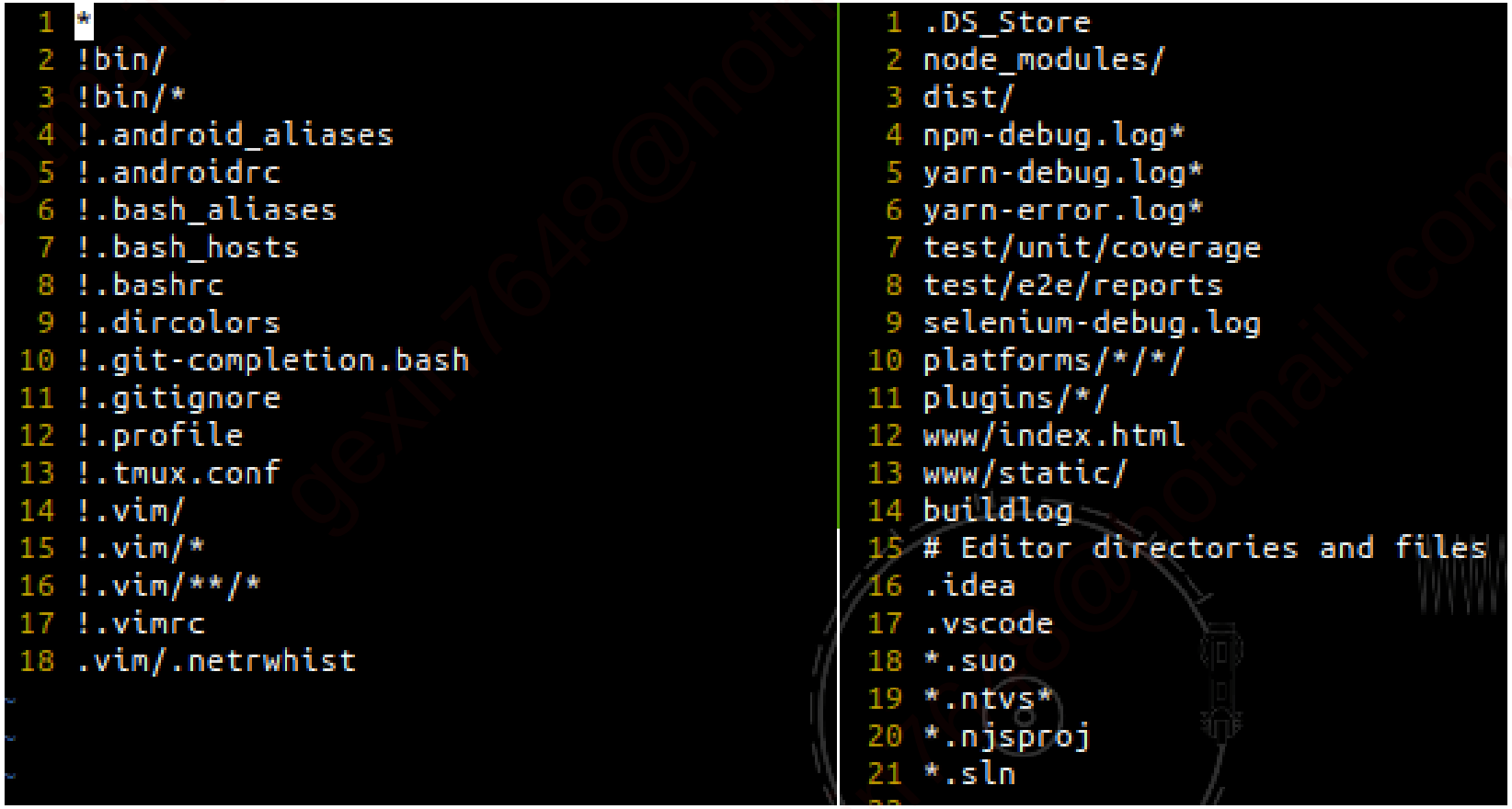
# Pull Request Merge Strategies



# Pull Request (PR) merge strategy – github flow



# Ask git to ignore files



```
*
!bin/
!bin/*
!.android_aliases
!.androidrc
!.bash_aliases
!.bash_hosts
!.bashrc
!.dircolors
!.git-completion.bash
!.gitignore
!.profile
!.tmux.conf
!.vim/
!.vim/*
!.vim/**/*
!.vimrc
.vim/.netrwhist
```

```
.DS_Store
node_modules/
dist/
npm-debug.log*
yarn-debug.log*
yarn-error.log*
test/unit/coverage
test/e2e/reports
selenium-debug.log
platforms/*/*/
plugins/*/
www/index.html
www/static/
buildlog
# Editor directories and files
.idea
.vscode
*.suo
*.ntvs*
*.njsproj
*.sln
```

# Group Exercise

Let us pair up, at least 2-3 people per group.
And let us run some exercises

1. Create a repo and add your team members into the repo
2. Make some feature branches and commit few changes (You can add file, folder or media etc.)
3. Everyone in the team pushes the changes to the team repo
4. Everyone pulls the latest repo
5. Everyone tries to modify the same file, make a commit and push to the repo
6. Pair together to fix the conflict and push
7. Pull the latest repo and revert one of your change

# Takeaways

- Never forget .gitignore
- Fix conflicts before PR
- Try to put all code for one function/subfunction/one bug fix in one git commit, which also means the commit should be small enough
- Try to make sure every commit will not break the build pipeline, which means it should pass code style check and unit tests.

# Homework / Challenges

https://github.com/JiangRenDevOps/DevOpsLectureNotesV6/blob/master/WK2_Git/git_exercise_1.md

https://github.com/JiangRenDevOps/DevOpsLectureNotesV6/blob/master/WK2_Git/git_exercise_2.md

https://learngitbranching.js.org/