Github与Git基础

内容概览:

- 通过一个互动式教程快速了解 Github 概念, 并学习 Github Repo, Github协作流和Issue.
- 学习 Git 基本概念, 使用Git的准备, Git基本操作, 与远程仓库交互, 分支管理这些Git基础知识.
- 通过互动教程持续打磨 Git 技能, 在本地使用Git命令重新完成Github协作流.
- 简要介绍Git的一些进阶知识, 包括提交规范, 高级功能和好用的工具.

快速入门

相信大家在平时都或多或少的使用过 Github 和 Git 进行代码管理. 我们首先通过一个互动式教程, 熟悉一下 Github 的最基本概念和操作.

https://github.com/skills/introduction-to-github (大约10分钟)

> 注: 这是一系列 Github 官方提供的互动式教程: https://skills.github.com/

> 官方文档: https://docs.github.com/zh/get-started/quickstart/hello-world

---

这个互动教程能学到什么:

- What is GitHub?
  - GitHub是一个协作平台，使用Git进行版本管理。GitHub是一个分享和贡献开源软件的流行场所。
- What is a repository?
  - 存储库是一个包含文件和文件夹的项目。存储库跟踪文件和文件夹的版本。
- What is a branch?
  - 分支是你的版本库的一个平行版本。分支允许你将你的工作与 main 分支分开。
- What is a commit?
  - 提交是对项目中的文件和文件夹进行的一系列修改。一个提交存在于一个分支中。
- What is a pull request?
  - 拉取请求将你的分支中的修改展示给其他人，并允许人们接受、拒绝或建议对你的分支进行额外的修改。
- What is a merge?
  - 合并将你的拉取请求和分支中的修改添加到主分支中。

> 注: 该互动教程是基于 Github Actions 实现的, 各个 Step 文档在 .github/steps 目录下.

完成这个互动教程之后, 我们来继续学习自己手动创建一个 Github 仓库, 在此基础上操作 Git 协作流.

Repo 与代码协作

在 Github 创建 Repo

任务: 在个人空间下创建一个名为 `test-repo` 的 repo.

> 可参考文档进行操作: https://docs.github.com/zh/get-started/quickstart/create-a-repo

协作流

GitHub Flow 是一个基于分支的轻量级工作流程。Github Flow

整个流程包括以下步骤:

- 创建Fork
- 创建分支
- 进行更改
- 创建拉取请求
- 解决审查评论
- 合并拉取请求
- 删除分支

基于 https://github.com/tannenbaumdev/demo-repo 演示一遍:

Fork -> Branch -> Commit -> Pull Request -> Review -> Merge

Github 除了基本的代码管理功能以外, 还包含社交属性. 下面我们来介绍 Issue 功能.

Issue与交流

https://docs.github.com/zh/issues/tracking-your-work-with-issues/quickstart

Github Issue 主要用来围绕代码做高效的交流讨论.

为什么要有Issue? 我的理解是, PR解决了What-How, 但没有说明Why; 利用Issue进行交流讨论, 对What-Why达成一致, 并最终通过PR解决问题.

经过以上学习, 我们已经初步了解了Github的代码管理和协作功能, 接下来我们开始学习 Github 的版本控制内核: Git.

Git 基础

基本概念

Git 是一种分布式版本控制工具. (文档: 关于版本控制)

什么是“版本控制”？版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。

为什么要做版本控制?

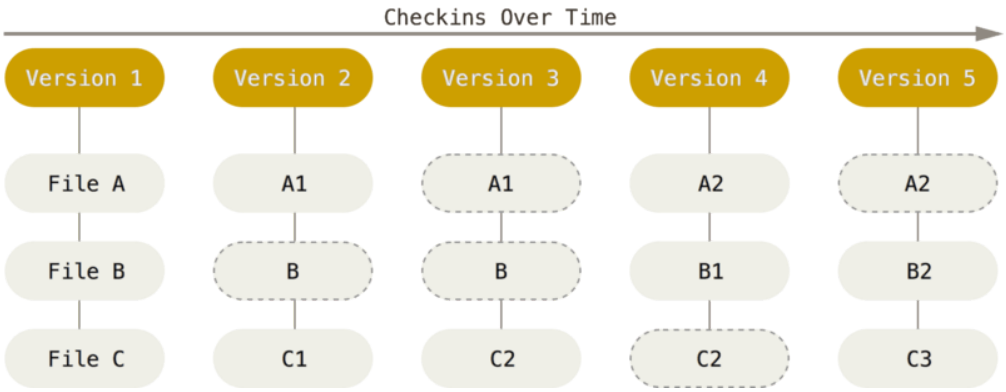
- 对比文件在各个状态的区别.
- 使文件回溯到某个指定的历史状态.

版本控制系统的演进:

本地版本控制系统 -> 集中化的版本控制系统 -> 分布式版本控制系统

主要特点

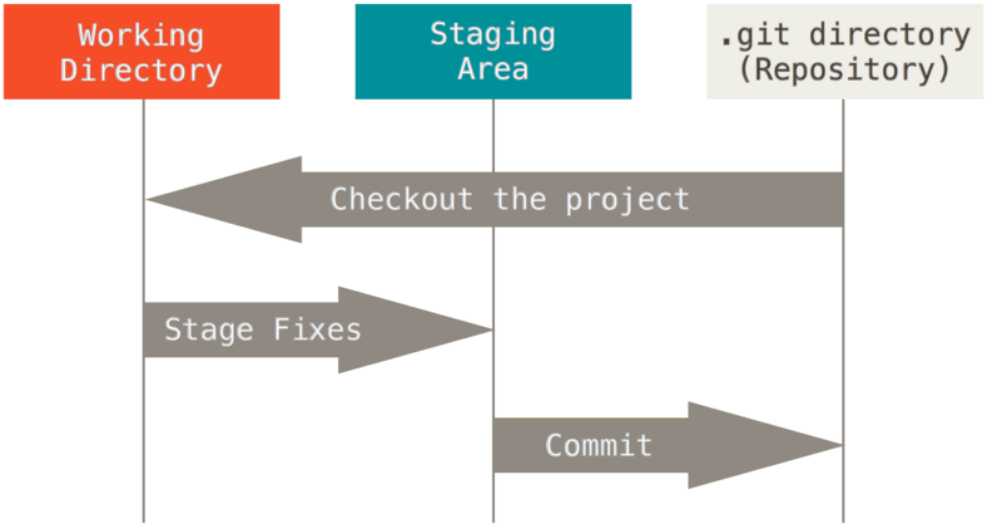
Git存储文件的方式: 存储文件快照而非文件差异.



Git文件的三种状态: **已提交（committed）**、**已修改（modified）** 和 **已暂存（staged）**。

- 已修改表示修改了文件，但还没保存到数据库中。
- 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中
- 已提交表示数据已经安全地保存在本地数据库中。

这会引出 Git 项目的三个 section：工作区、暂存区以及 Git 目录。



- 工作区是对项目的某个版本独立提取出来的内容。 这些从 Git 仓库的压缩数据库中提取出来的文件，放在磁盘上供你使用或修改。
- 暂存区是一个文件，**保存了下次将要提交的文件列表信息**，一般在 Git 仓库目录中。
- Git 仓库目录是 Git 用来保存项目的元数据和对象数据库的地方。
  - 这是 Git 中最重要的部分，从其它计算机克隆仓库时，复制的就是这里的数据。

基本的 Git 工作流程如下：(注意与之前提到的Github Flow区分开)

1. 在工作区中修改文件。
2. 将你想要下次提交的更改选择性地暂存，这样只会将更改的部分添加到暂存区。
3. 提交更新，找到暂存区的文件，将快照永久性存储到 Git 目录。

接下来我们要开始使用Git了, 在这之前先做一点准备工作.

准备工作

首先需要在本地安装 git 工具. 参考 安装 Git

确认本地 git 已安装好:

```
> git --version
git version 2.20.1
```

之后需要做一些基本配置, 这里我们主要配置 **全局 user 信息**.

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

通常我们会用 SSH 协议访问 Git 仓库, 使用 SSH 协议还需要创建一对密钥, 并将公钥添加到 Github 上.

参考文档: 生成新的 SSH 密钥并将其添加到 ssh-agent

本地生成 SSH 密钥:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

将密钥添加到 ssh-agent:

```
ssh-add ~/.ssh/id_ed25519
```

最后, 将 SSH 公钥添加到 GitHub 上的帐户: 新增 SSH 密钥到 GitHub 帐户

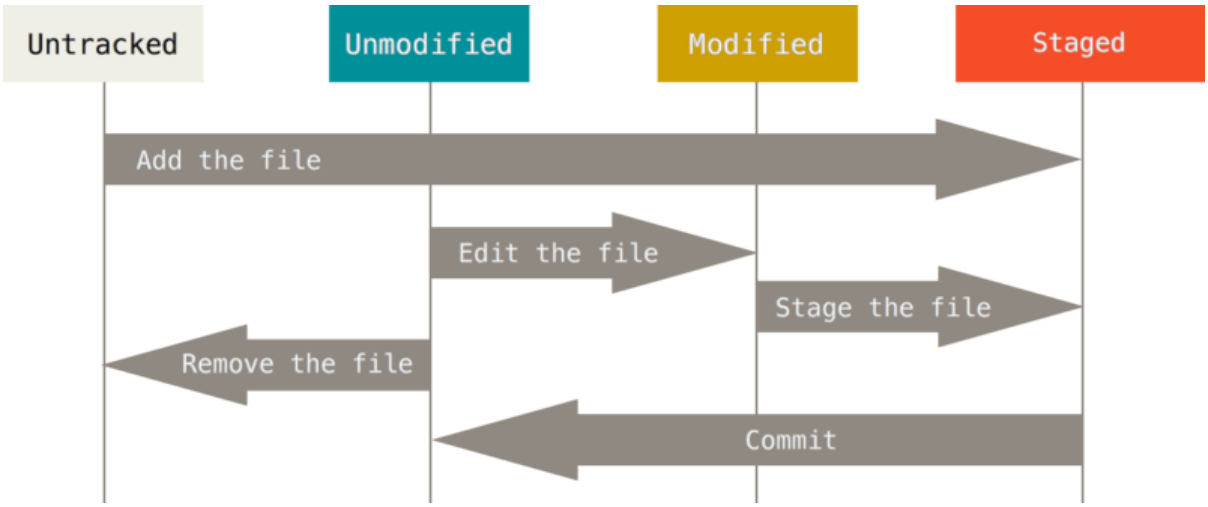
做好以上配置后, 就可以开始学习 Git 基本操作命令了.

基本操作

使用 `git clone` 命令克隆远程仓库到本地.

```
$ git clone https://github.com/tannenbaumdev/demo-repo.git
```

下面我们要学习 Git 如何处理文件更新操作. 首先要知道在 Git 中一个文件的状态变化周期:



Untracked 是未跟踪状态, 其他都是已跟踪状态 (已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录).

对于一个刚刚 clone 到本地的 repo, 所有文件都是 Unmodified 状态.

使用 `git status` 命令查看文件状态:

```
$ git status
On branch main
Your branch is up-to-date with 'origin/main'.
nothing to commit, working directory clean
```

此时我们新创建一个文件:

```
$ echo 'hello world' > a.md
```

再次执行 `git status`, 系统提示我们存在 Untracked files:

```
$ git status
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        a.md

nothing added to commit but untracked files present (use "git add" to track)
```

使用命令 `git add` 开始跟踪一个文件, 或者将一个 Modified 的文件提交到暂存区, 变为 Staged 状态.

```
$ git add a.md
$ git status
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   a.md
```

执行 `git commit` 命令, 提交暂存区的文件.

```
git commit
```

几个常用的commit参数:

- `-m` 添加 commit message
- `--amend` 追加提交
- `-a` 跳过暂存直接提交

提交之后, 使用 `git log` 命令可以查看提交历史.

常用的log参数:

- `-p` 显示每次提交所引入的差异
- `-2` 最近2次
- `--oneline` 每个commit用1行显示

如果我们在 `git add` 之后不想提交, 想把修改从 Staged 状态撤销, 可以用 `git reset` 命令.

以上就是常用的 git 基础命令了, 下面我们再来介绍一下本地如何与远程仓库交互.

远程仓库交互

我们本地的代码是通过 `git clone` 从远程拉取的, 使用 `git remote` 命令可以查看远程仓库.

```
$ git remote
origin
```

使用 `git remote add` 添加其他远程仓库.

```
$ git remote add myorigin https://github.com/a/b
```

使用 `git fetch` 拉取远程仓库代码.

```
$ git fetch origin
```

> 注: pull = fetch + merge

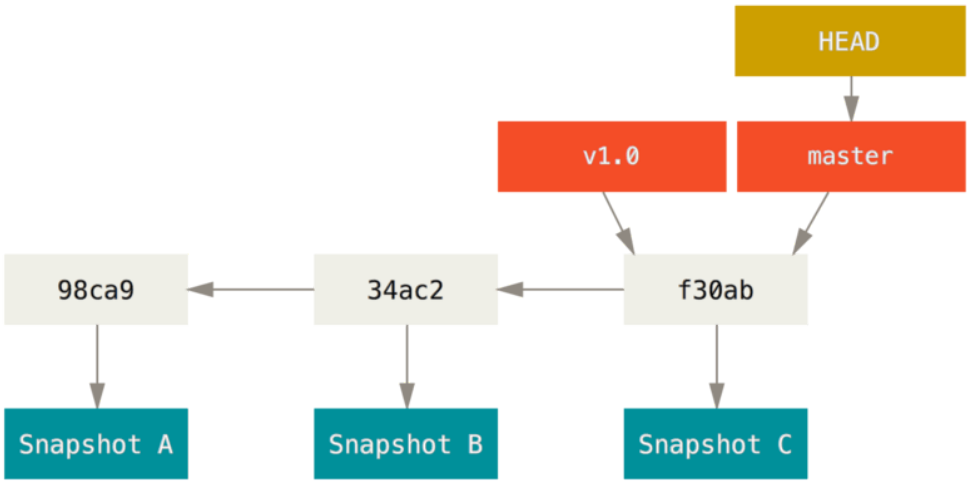
接下来介绍Git最重要的特性 (可能没有之一): 分支.

分支

> 小贴士: 这个可视化工具可以方便地学习分支变化. https://learngitbranching.js.org/?locale=zh_CN

"有人把 Git 的分支模型称为它的“必杀技特性”，也正因为这一特性，使得 Git 从众多版本控制系统中脱颖而出。" 分支简介

Git 的分支非常轻量，原因是它本质上仅仅是指向提交对象的可变指针。



使用 `git branch` 命令创建分支.

```
$ git branch dev-1
```

这会在当前所在的提交对象上创建一个指针.

那么 Git 又是怎么知道当前在哪一个分支上呢? 也很简单, 它有一个名为 `HEAD` 的特殊指针, 指向当前所在的本地分支.

使用 `git checkout` 命令切换分支.

```
$ git checkout dev-1
```

> 注: 使用 git checkout -b 创建并切换分支.

使用 `git merge` 命令合并分支. (记得先在dev-1分支上面做一些改动并提交)

```
$ git checkout main
$ git merge dev-1
```

这时我们会发现并没有所谓的"合并", 而是 main 指针向前移动到 dev-1 指针位置, 这种叫作 fast-forward merge.

> 如果顺着一个分支走下去能够到达另一个分支，那么 Git 在合并两者的时候， 只会简单的将指针向前推进（指针右移），因为这种情况下的合并操作没有需要解决的分歧——这就叫做 “快进（fast-forward）”。

如果不能 fast-forward merge, 就会变成 typical merge. (演示)

> 注: 即使满足 fast-forward 条件, 也可通过指定 `--no-ff` 参数转为 typical merge.

如果合并过程中存在冲突, 需要解决冲突, 然后执行 `git merge continue` 继续完成合并.

远程分支 相关操作请通过文档自学.

其他命令 (自学)

- stash
- rebase 暂时不用, 因为 Merge 已经能解决大多数问题了.
- revert
- reset
- reflog
- tag

Git实践

互动练习

这里介绍一套互动式Git实践教程, 非常适合用来学习Git常用操作: Git Exercises

我们从头开始做一两个练习体验下.

再看协作流

学习了 Git 命令以后, 我们重新尝试在本地用 Git 命令完成之前的协作流.

Fork -> Clone Origin -> Add Remote -> Branch -> Commit -> Pull Request -> Merge

基于 https://github.com/tannenbaumdev/demo-repo 演示.

最后我们简要了解一些 Git 进阶知识.

Git进阶

提交规范

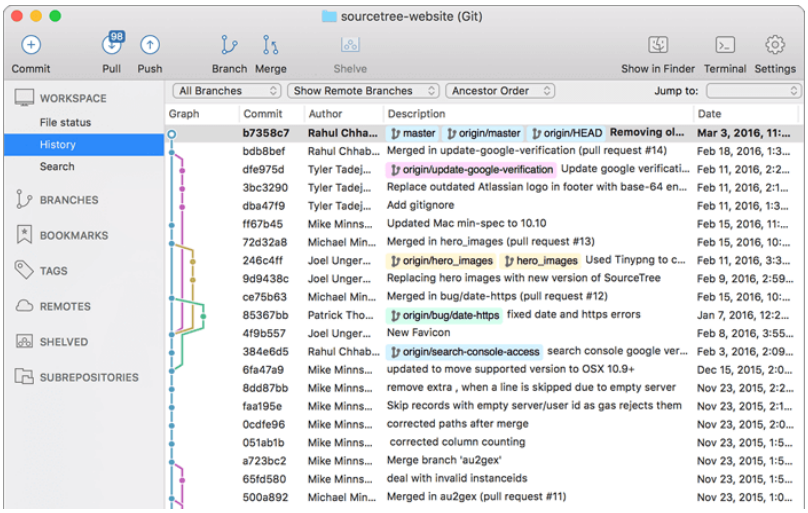
约定式提交 :让提交信息清晰易懂, 便于检索

高级功能

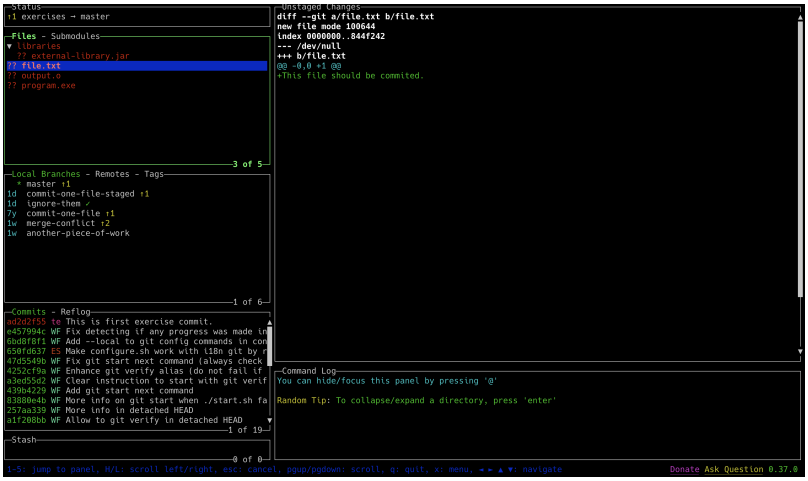
- git hook :特定阶段触发自定义脚本
- git submodule :项目依赖
- git LFS :解决大文件存储问题

Git小工具

GUI: SourceTree



TUI: lazygit



SQL on Git: gitql

```
gitql> select * from branches
+------------------------+-----------------------------------+------------------------------------------+
|          name          |             full_name             |                   hash                   |
+------------------------+-----------------------------------+------------------------------------------+
| another-piece-of-work  | refs/heads/another-piece-of-work  | b15f3a7d657c407d1fbe1e081f8c6d3e790a74ef |
+------------------------+-----------------------------------+------------------------------------------+
| commit-one-file        | refs/heads/commit-one-file        | 9abacd8485e8ca5bce003f1745f3eb4f422a64e9 |
+------------------------+-----------------------------------+------------------------------------------+
| commit-one-file-staged | refs/heads/commit-one-file-staged | 0d019e8d09dcdeb138cf699ee2e1c6fadc30de1b |
+------------------------+-----------------------------------+------------------------------------------+
| ignore-them            | refs/heads/ignore-them            | 11edd38542b6ec7f13975bb7de12bb831d3a7bc1 |
+------------------------+-----------------------------------+------------------------------------------+
| master                 | refs/heads/master                 | ad2d2f558e02030aa7888e5057f01bc1fd04555d |
+------------------------+-----------------------------------+------------------------------------------+
| merge-conflict         | refs/heads/merge-conflict         | 36f46ef883b1fa57e8cde5e108962ba439f0dc68 |
+------------------------+-----------------------------------+------------------------------------------+
```

扩展学习

- Git基础命令速查
- Pro Git 中文版
- GitHub Student Developer Pack
- Git Katas
- Git Immersion

- Github Discussions 文档

练习与总结

1. 在个人空间下, 完成 github-starter-course 互动课程.
2. 按照 `协作流` 的操作流程, 为 pull-request-test 提交一个 Pull Request, 要求:
    1. 提交内容: 在root目录下, 创建一个名为 `dev-你的username.md` 的文件, 文件内容为空.
    2. 分支名为 `dev-你的username`, Pull Request名称与分支名相同.
    3. 解决Reviewer提出的修改意见.
    4. 分支合并到main.
3. 完成 Git Exercises 剩余练习.
4. 尝试学习 Git内部原理相关文档.
5. 在 Discussions 中, 创建一个 `Show and tell` 类型的 Discussion, 命名为 `Devtool Note (你的username)` (例如 `Devtool Note (teckick)`), 创建完成后, 对本次学习内容进行总结, 写到 Comments 中 (中英文均可).