Git提交规范

没啥用的导入

什么是 Git、GitHub?

Git

Git（读音为/gɪt/）是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。 也是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

GitHub

GitHub 是一个面向开源及私有软件项目的托管平台，因为只支持 Git 作为唯一的版本库格式进行托管，故名 GitHub。GitHub 拥有 1 亿以上的开发人员，400 万以上组织机构和 3.3 亿以上资料库。

为什么要用 Git

在开发项目的时候，我们可能会不断地去修改代码，但是有时候会遇到，想查看某一时间的代码这种情况，如果没有版本控制器，你可能需要不断地定时备份代码，但这样显然是很麻烦的，而且备份也不一定好用，比如某个时间点并没有修改代码，那么备份就重复了；再比如虽然备份了代码，但你并不知道两个版本有什么区别。

Git 的安装和 GitHub 仓库创建

Git 安装

> 已经安装的可跳过

Git 安装配置 | 菜鸟教程

Git 配置

> 已经能正常使用的跳过

```
git --version
git config --global user.name "你的GitHub用户名"
git config --global user.email 你的GitHub邮箱
```

GitHub 仓库创建

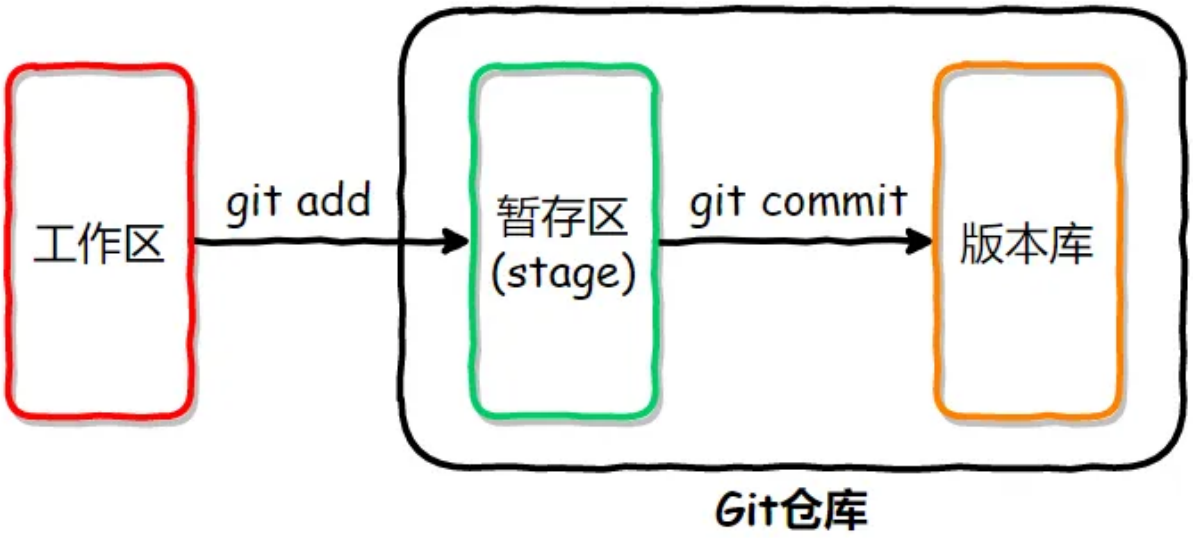
> 这一步请在个人账户中操作，不要影响到战队存储库

Git 创建仓库 | 菜鸟教程

部分概念理解

工作区、暂存区、版本库

- 工作区 = 当前正在改的文件
- 暂存区 = 准备要保存的文件
- 版本库 = 正式保存的版本



> **注意：**
>
> 1. 在 Git 中，必须明确地将文件添加到暂存区，然后才能将其提交到版本库中。

Git 管理的第一原则

1. 每天干活前做的第一件事：`git pull main` 拉取最新代码
2. 频繁小步提交：尽可能将 Git 更改精确到代码的每一次改动
3. 提交信息要有意义：简洁明了，让人一眼知道这一个版本改动了什么

Git 的常用操作和相关规范

> - 以下操作基于`bash`，相应`VSCode`操作自行搜索
> - 使用仓库为 2024 赛季步二仓库

创建仓库

不解释了，应该看得懂

```
# 创建本地仓库所在文件夹
mkdir 想要创建的仓库名

# 初始化仓库
cd 刚刚创建的文件夹
git init

# 随便创建一个文件写点东西
echo "README" > README.md

# 将要提交的文件添加进暂存区
git add README.md
# 或者一次性把整个仓库的文件添加进暂存区（少用）
git add .
# 如果要删除暂存区的文件可以
git rm 要删除的文件

# 提交第一个版本
git commit -m "要推送的消息"
# 如果推送消息错了可以
git commit --ament -m "新的推送消息"
```

克隆远程仓库

```
git clone https://github.com/FZSDRM/infantry_2.git
```

创建分支

通常会使用到的分支有`master/main`、`develop`、`feature`、`test`、`release`、`hotfix`六个分支，目前我们打算使用的分支有`main`、`dev`（对于多个部分的开发分支以以下命名方式为例：`dev_chassis`、`dev_gimbal`）、`feature`（命名可以为`feat_od_wheel`、`feat_dm_motor`等，名字不要太长）、`hotfix`，不同分支的规范如下：

1. `main`：仓库的主分支，是确保能稳定使用的版本。

2. `dev`：开发分支，每个开发人员各自负责一个分支，在分支合并窗口提交最稳定的分支版本提交审查合并。
3. `feature`： `dev` 分支的子分支，每个新功能应单独创建 `feature` 分支，功能基本稳定后合并入对应 `dev` 分支。
4. `hotfix`：大小比赛前应创建的分支，用于应对临场出现的 bug 的紧急修复。赛后应将代码审查优化后合并入对应 `dev` 或 `main` 分支。

```
# 创建新分支
git branch dev
# 切换分支
git checkout dev

# 也可一步创建并切换到新分支
git checkout -b feature

# 如果想在父分支上创建子分支，有两种操作
# 1
git checkout 父分支
git checkout -b 子分支
# 2
# 有点抽象，就找到一篇文章讲了，git官方文档没看到在哪，就不讲了
```

切换到新分支后后续所有的代码提交都是上传到对应这个分支，直到下一次切换分支。

本地重命名分支

建议少干改名的事，想好再创建。

```
git branch -m <old-branch-name> <new-branch-name>
```

删除分支

```
# 场景：删除分支
git branch -d <branch-name>
# 或
git push origin :<branch-name>

# 场景：强制删除未合并的分支
git branch -D <branch-name>
```

抓取分支

```
# 拉取所有远程分支并更新本地分支
```

```
git fetch --all

# 拉取一个特定的远程分支到本地
git fetch origin <branch-name>

# 在本地创建基于远程分支的新分支
git checkout -b <new-branch-name> origin/<remote-branch-name>

# 拉取远程分支并自动与本地分支关联（--track可缩写为-t）
git checkout --track origin/<remote-branch-name>
```

推送分支

```
# 推送当前分支到远程仓库，并与远程分支关联
git push -u origin <branch-name>

# 推送当前分支到远程仓库，并与远程分支合并
git push origin <branch-name>

# 强制推送当前分支到远程仓库
git push -f origin <branch-name>
```

分支回退

```
# 指向上一次提交，但不删除此次提交
git reset HEAD~1

# 删除此次提交，撤回到上一次提交
git reset --hard HEAD~1
```

分支合并

将指定分支合并到当前分支：

```
git merge <branch_name>
```

冲突处理

当两个分支上的代码修改了同一部分，并且尝试将这两个分支合并时，就会发生代码冲突。Git 提供了以下步骤来解决冲突：

- 运行 git status 命令查看哪些文件包含冲突。
- 编辑有冲突的文件，手动解决文件中的冲突。

- 对编辑后的文件进行 git add，标记为已解决冲突的文件。
- 使用 git commit 提交更改，Git 会自动生成一个合并提交，其中包含各自分支中的更改。

> 注意：在解决冲突前，最好先备份当前的代码状态，以免不小心破坏代码库。另外，在处理冲突之前，可以通过运行 git diff 命令来查看冲突的源代码，以便更好地理解要解决的问题。

代码审查

团队内部审查

这还要想？直接去线下真实就行了！干他丫的敢这么写代码！

其他团队开发者审查

> 参考：Git - 对项目做出贡献、GitHub pull request 入门（图解+原理+git 命令+可能有用的经验） - 知乎和在 GitHub 上玩转开源项目的 Code Review - 胡说云原生 - 博客园

这部分代码过于多且繁杂了，可以自己看官方文档提供的方式。总的来说就是：

> 此处默认 clone 的分支叫 `dev`，创建的新分支为 `devdev`

1. `fock` 要审查的仓库分支
2. `clone` 已经 `fock` 完的仓库到本地（此时这个仓库应该是在你自己的账号下）
3. 与原项目仓库建立连接：

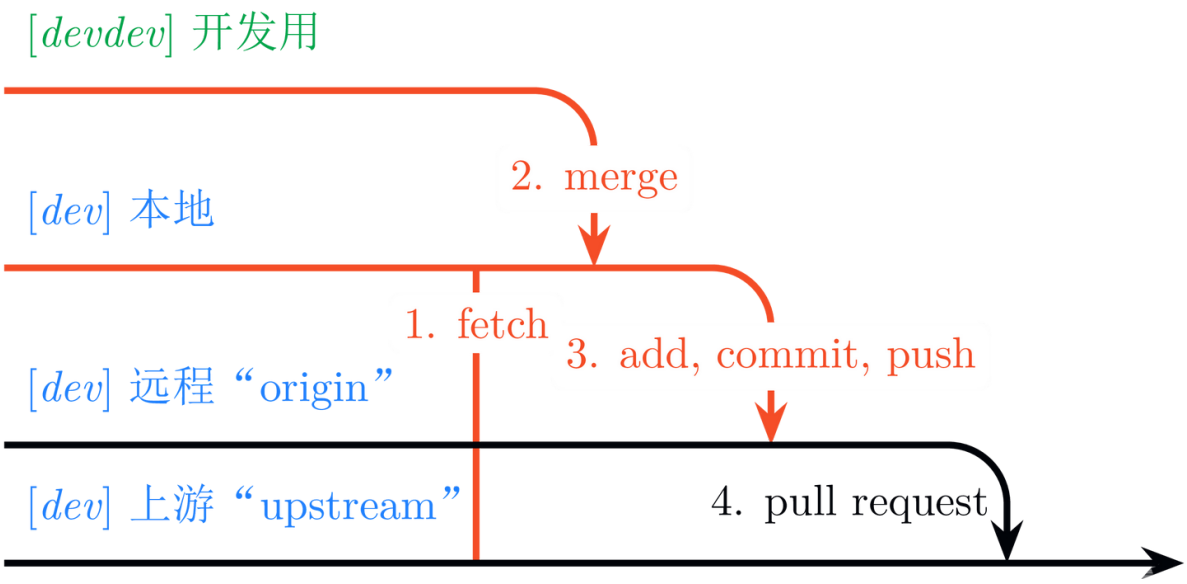
```
git remote add upstream <原项目的Code里复制的地址>
```

4. 建立本地开发分支

```
# 以下二选一执行
# git switch是2.23版本新加入的，老版本请用git checkout。
git switch -c devdev  # 新建并切换到devdev分支
git checkout -b devdev
```

5. 修改代码

6. 提交代码



7. 与原项目同步

```
# 默认本地clone分支为dev
git fetch upstream dev
```

8. 合并自己的代码

```
git switch dev  # 确保切换到dev分支
git merge devdev
```

9. git add & git commit & git push

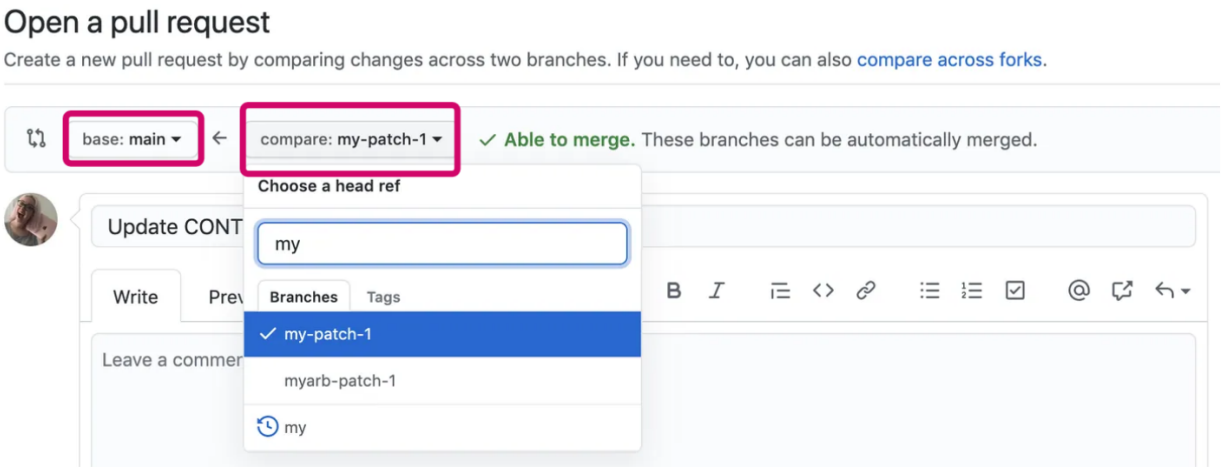
> 这一部分和提交自己的项目到 GitHub 完全相同，只需要注意原项目是否有 commit 指南即可。

```
git add .
git commit -m 【需要的消息内容或格式】
git push origin dev
```

10. pull request

> 只有这一步只能网页操作

直接点击上述页面中的 Contribute 中 Open pull request 即可：



> 能先提出 issue 的话，**先写一个 issue，说明一下有某某问题，我觉得可以如何做之类，看作者的回复**。说不定作者已经在改进这里，而其他人给的方案通常是不如作者的。

推送消息要求

其他

参考 `.gitignore` 文件

```
# 忽略build文件夹下所有文件
build/
# 其他文件、文件夹视情况忽略，建议上传
```

参考推送消息

```
feat: 底盘新增全向轮解算
fix:  解决遥控Z键疯车问题
docs: 完善modules/super_cap文档解释
perf: 更改开平方根算法为卡马克快速开平方根算法

revert: 因chassis部分错误过多回滚到上一个版本

refactor: 其他更改
```

其中：

1. feat: 新增功能
2. fix: 修复 bug
3. docs: 仅文档更改
4. perf: 改进性能的代码更改

5. revert：回滚到上一个版本
6. refactor: 既不修复 bug 也不添加特性的代码更改（仅这一类型允许不详细提及代码更改）

如有必要也可使用其他类型。

> **注意：**
>
> - 提交问题必须为同一类别。
> - 提交问题不要超过 3 个。
> - 提交的 commit 发现不符合规范，`git commit --amend -m "新的提交信息"` 或 `git reset --hard HEAD` 重新提交一次。

一点小建议

1. 除了 `VSCode` 建议的插件外，建议多装一个 `Git Graph` 插件，会更方便把握整个仓库分支的结构。
2. 安装 `Github Copilot` 插件：知道你们不爱写注释也不爱干那些琐碎的事，不如让 AI 先帮你写一部分然后再自己根据要求修改。在 GitHub 上申请一个学生认证之后就能免费用了，四五天左右生效。
3. 多帮忙审一审其他机器人的代码（该赛季所有代码审查权限是对所有团队全部开放的），可以学习学习其他人解决问题的思路，顺便也能帮忙看看有没有什么问题（应该也没有人想手头有事但是又不得不跑去实验室修 bug 吧）。

参考资料

- Git Tutorial --- 最通俗易懂的教程，没有之一-CSDN 博客
- Git 使用教程：最详细、最正宗手把手教学（万字长文） git 教程-CSDN 博客
- Git 官方文档
- Git 小白速成指南 - 从零开始学会版本控制