基础知识

先看一下这个（git使用简明教程）https://www.bootcss.com/p/git-guide/

git与github/gitlab的关系

1. Git - 版本控制的核心

- **是什么？** Git 是一个**分布式版本控制系统**。它是由 Linus Torvalds（Linux 之父）为了管理 Linux 内核开发而创建的。
- **核心功能：**
  - **版本控制：** 记录每一次文件的改动，你可以回退到任何一个历史版本。
  - **分支管理：** 可以创建分支，在不影响主线（主分支）的情况下开发新功能，完成后再合并回主线。
  - **协作：** 每个开发者都拥有完整的代码仓库副本，可以独立工作。
- **关键点：** Git 是一个**命令行工具**，在你的本地计算机上运行。它本身不提供网页界面或托管服务。

2. GitHub - 基于 Git 的代码托管社区

- **是什么？** GitHub 是一个**基于 Git 的代码托管平台**。它是全球最大的开源社区。
- **核心功能：**
  - **远程仓库托管：** 为你提供一个在云端的服务器，用来存放你的 Git 仓库。这样你的代码就不会因为本地电脑损坏而丢失，团队成员也可以轻松访问。
  - **协作功能：** 提供了强大的协作工具，如：
    - **Pull Request：** 请求将他人的代码合并到主分支，并在此过程中进行代码审查和讨论。
    - **Issue：** 用于跟踪任务、功能请求和 Bug。
    - **Wiki：** 为项目编写文档。
  - **社交化：** 你可以关注其他开发者，给项目点赞。
- **商业模式：** 公有仓库免费，私有仓库需要付费。**主要是一个 SaaS（软件即服务）平台**，你使用它提供的网站和服务。

3. GitLab - 更全面的 DevOps 平台

- **是什么？** GitLab 也是一个**基于 Git 的代码托管平台**，但它的目标是成为一个覆盖软件开发生命周期全过程的 **“一站式 DevOps 平台”**。
- **核心功能：** 它包含了 GitHub 的所有核心功能（仓库托管、Pull Request、Issue 等），并在此基础上增加了大量**开箱即用**的强大功能：
  - **CI/CD：** 这是 GitLab 的王牌功能。你可以在仓库中直接编写一个 `.gitlab-ci.yml` 文件，来定义自动化构建、测试、部署的流程。无需集成第三方工具。
  - **内置容器注册表：** 可以方便地存储和管理 Docker 镜像。
  - **内置 Kubernetes 集成：** 轻松将应用部署到 K8s 集群。
  - **安全扫描：** 自动进行依赖项漏洞扫描、代码安全扫描等。
  - **Wiki 和 Pages：** 同样具备文档和静态网站托管功能。
- **部署方式：** 这是与 GitHub 的一个巨大区别。GitLab 提供多种选择：
  - **SaaS：** 类似 GitHub，使用 gitlab.com 官方服务。
  - **自托管：** 你可以将 GitLab 部署到自己的服务器上（无论是公司内网还是私有云），实现**完全的私有化控制**。社区版（CE）免费，功能强大的企业版（EE）收费。

GitLab 完整工作流程（团队协作版）

阶段一：准备工作

1. 获取代码到本地

```
# 克隆远程仓库到本地
git clone https://gitlab.com/你的用户名/你的项目名.git
```

- **远程仓库**：GitLab 服务器上的官方代码库，团队共享
- **本地仓库**：你电脑上的完整代码副本，包含所有历史记录
- **克隆后**：你的电脑上就有了完整的项目，可以独立工作

2. 确认工作分支

```
# 查看当前所在分支
git branch
```

```
# 查看远程仓库信息
git remote -v
```

阶段二：开始新功能开发

1. 基于 Issue 创建新分支！

最佳实践：每个新功能都对应一个 Issue

```
# 从最新的main分支创建功能分支
git checkout main
git pull origin main                          # 确保本地main是最新的
git checkout -b feature-用户登录功能      # 创建并切换到新分支
```

- **为什么要在分支上工作？** 避免直接修改主分支，如果新功能有问题，不会影响主分支的稳定性
- **分支包含原有代码吗？** 是的！新分支创建时是主分支的完整副本，你在此基础上添加新代码

2. 日常开发工作循环

这是你每天的开发节奏：

```
# 1. 修改代码（在编辑器中工作）
# 2. 查看改了哪些文件
git status

# 3. 将改动添加到暂存区
git add .                                     # 添加所有改动
# 或
git add 文件名                                # 只添加特定文件

# 4. 提交到本地仓库
git commit -m "feat: 实现用户登录接口"
```

🔍 **详细解释：**

- `git add`：把改动"打包"，准备提交
- `git commit`：把打包的改动"封箱贴标签"，保存到本地历史记录
- **此时**：改动只在你电脑上，GitLab 上还看不到（因为这是在本地仓库上的操作）

3. 与团队同步

```
# 推送分支到远程仓库（GitLab）
git push origin feature-用户登录功能

# 如果远程分支不存在，第一次推送需要：
git push --set-upstream origin feature-用户登录功能
```

🔍 **详细解释：**

- `git push`：把你的本地提交"寄出去"到 GitLab
- **现在**：团队其他人能在 GitLab 上看到你的分支和代码了

阶段三：代码审查与合并

1. 创建合并请求（Merge Request）

- 在 GitLab 网页端，你的分支推送后会看到创建 MR 的提示
- 或者手动：进入项目 → Merge Requests → New Merge Request

填写 MR 信息：

- **源分支**：你的功能分支 `feature-用户登录功能`
- **目标分支**：`main`（要合并到的主分支）
- **标题**：清晰描述这个 MR 的目的
- **描述**：详细说明改了什么，为什么改
- **分配审查者**：选择团队成员审查你的代码
- **关联 Issue**：链接到对应的 Issue（重要！）

2. 代码审查流程

审查者会：

- 阅读代码，提出建议
- CI/CD 流水线自动运行测试
- 讨论修改点

你可能需要：

```
# 根据审查意见修改代码后
git add .
git commit -m "fix: 根据审查意见修复xx问题"
```

```
git push origin feature-用户登录功能     # 推送到同一分支，MR会自动更新
```

3. 合并与清理

审查通过后：

- Maintainer 点击 "Merge" 按钮
- 你的代码就正式进入主分支了

合并后清理：

```
# 切换回主分支
git checkout main

# 拉取最新代码（包含你刚合并的功能）
git pull origin main

# 删除本地功能分支（可选）
git branch -d feature-用户登录功能

# 删除远程功能分支（可选）
git push origin --delete feature-用户登录功能
```

阶段四：处理团队其他人的更新

当别人也提交了代码时：

```
# 定期从主分支拉取最新代码
git checkout main
git pull origin main

# 如果在功能分支上，想同步主分支的更新：
git checkout feature-你的功能分支
git merge main
# 或使用变基（更推荐）
git rebase main
```

实用命令清单

```
```

```
# 查看状态
git status
git log --oneline

# 分支操作
git branch -a                        # 查看所有分支
git checkout 分支名                  # 切换分支
git branch -d 分支名                 # 删除分支

# 撤销操作
git reset --soft HEAD~1              # 撤销上次commit，保留改动
git restore 文件名                   # 丢弃文件的未暂存改动
```