Git 基本使用（完整版）

目录

简介

Git 是目前最流行的分布式版本控制系统，由 Linux 的创始人 Linus Torvalds 于 2005 年开发。它主要用于跟踪文本文件（如源代码）的更改，方便多个开发者协同工作。

Git 的安装

在 Ubuntu 22.04 上安装 Git

打开终端，更新软件包列表：

```
sudo apt update
```

安装 Git：

```
sudo apt install git
```

验证安装是否成功：

```
git --version
```

在 Windows 上安装 Git

1. 访问 Git for Windows 官方网站。
2. 下载最新的安装程序。
3. 运行安装程序，按照默认设置完成安装。
4. 安装完成后，打开命令提示符或 Git Bash，输入 `git --version` 验证安装。

在 macOS 上安装 Git

使用 Homebrew 安装

如果已安装 Homebrew，可以在终端输入：

```
brew install git
```

使用安装包安装

1. 访问 Git 官方下载页面。
2. 下载最新的安装程序并运行。
3. 安装完成后，在终端输入 `git --version` 验证安装。

Git 的基本原理

版本控制的概念

版本控制是一种记录一个或多个文件内容变化，以便将来查阅特定版本修订情况的系统。它允许多人协作开发，跟踪和管理文件的历史版本。

Git 的工作机制

- **工作区（Working Directory）**：你在电脑上能看到的目录。
- **暂存区（Staging Area）**：用于临时存放你的改动。
- **本地仓库（Repository）**：通过 `git commit` 将暂存区的内容保存到本地仓库。
- **远程仓库（Remote Repository）**：托管在网络上的 Git 仓库，如 GitHub、GitLab。

Git 与其他版本控制系统的比较

- **分布式 vs 集中式**：Git 是分布式的，每个开发者的本地仓库都是完整的。
- **性能**：Git 速度快，效率高，特别是在处理大型项目时。
- **分支管理**：Git 的分支操作轻量级且灵活。

Git 的基本配置

`git config` 配置 Git

在使用 Git 之前，需要配置你的用户信息，这些信息会包含在每次提交中。

设置全局用户名和邮箱

```
git config --global user.name "你的姓名"
git config --global user.email "你的邮箱地址"
```

- `user.name`：设置提交者的姓名。
- `user.email`：设置提交者的邮箱。

例如：

```
git config --global user.name "Li Hua"
git config --global user.email "lihua@example.com"
```

查看当前配置

```
git config --list
```

编辑配置文件

您可以直接编辑配置文件来修改设置：

- 全局配置文件位置：`~/.gitconfig`
- 项目配置文件位置：`项目目录/.git/config`

配置别名

为常用命令设置别名，方便使用：

```
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
```

现在，你可以使用 `git st` 来替代 `git status`。

Git 的基本指令

`git init` 初始化仓库

在当前目录新建一个 Git 仓库：

```
git init
```

`git clone` 克隆仓库

克隆远程仓库到本地：

```
git clone <repository_url>
```

`git add` 添加文件到暂存区

添加单个文件：

```
git add filename
```

添加所有更改的文件：

```
git add .
```

`git commit` 提交更改

将暂存区的文件提交到本地仓库：

```
git commit -m "提交说明"
```

`git status` 查看状态

查看工作区和暂存区的状态：

```
git status
```

`git log` 查看提交历史

查看提交记录：

```
git log
```

分支管理

`git branch` 创建分支

创建新分支：

```
git branch branch_name
```

查看所有分支：

```
git branch
```

`git checkout` 切换分支

切换到指定分支：

```
git checkout branch_name
```

`git merge` 合并分支

将分支 `branch_name` 合并到当前分支：

```
git merge branch_name
```

`git switch` 切换或创建分支

Git 2.23 引入了 `git switch` 命令：

- 切换到已有分支：

```
git switch branch_name
```

- 创建并切换到新分支：

```
git switch -c new_branch_name
```

查看差异

`git diff` 查看更改

- 查看工作区与暂存区的差异：

```
git diff
```

- 查看暂存区与本地仓库的差异：

```
git diff --staged
```

远程仓库操作

`git remote` 管理远程仓库

查看已添加的远程仓库：

```
git remote -v
```

添加远程仓库：

```
git remote add origin <repository_url>
```

`git push` 推送到远程仓库

将本地分支推送到远程仓库：

```
git push origin master
```

`git pull` 拉取远程更新

从远程仓库获取并合并更新：

```
git pull origin master
```

`git fetch` 获取远程更新

只获取远程仓库的更新，不进行合并：

```
git fetch origin
```

实验：Git 实践操作

实验目标

通过本实验，您将：

- 熟悉 Git 的基本命令和操作流程。
- 能够在本地创建 Git 仓库，添加和提交文件。
- 理解并实践分支的创建、切换和合并。
- 掌握将本地仓库推送到远程仓库的流程。
- 学会配置 Git 的用户信息和别名。

完成指标：

- 成功安装并配置 Git。
- 初始化一个本地 Git 仓库并进行基本操作。
- 创建并合并分支。
- 将本地仓库推送到远程仓库。
- 能够查看并理解 Git 的状态和日志信息。

实验准备

- **环境要求**：
  - 一台已安装 Git 的计算机（Ubuntu 22.04/Windows/macOS 均可）。
  - 访问互联网的能力（用于访问远程仓库）。
- **账号要求**：
  - 已注册 GitHub、GitLab 或其他 Git 托管服务账号。
- **知识要求**：
  - 基本的命令行操作知识（如创建目录、切换目录、编辑文件等）。

实验步骤

步骤一：安装和配置 Git

1. **安装 Git**：

   根据您的操作系统，按照上述安装指南安装 Git。

2. **配置用户信息**：

   设置全局用户名和邮箱，这些信息会出现在您的提交记录中。

   ```
   git config --global user.name "Your Name"
   git config --global user.email "youremail@example.com"
   ```

   注意：请将 `"Your Name"` 和 `"youremail@example.com"` 替换为您的实际姓名和邮箱。

3. **验证配置**：

   ```
   git config --list
   ```

   检查是否包含您刚才设置的用户名和邮箱。

4. **配置别名（不建议）**：

   为常用命令设置简短的别名，但是初学阶段，建议大家不要打别名。

   ```
   git config --global alias.st status
   git config --global alias.co checkout
   git config --global alias.br branch
   git config --global alias.ci commit
   ```

步骤二：初始化本地仓库

1. **创建项目目录**：

   ```
   mkdir git-lab
   cd git-lab
   ```

2. **初始化 Git 仓库**：

   ```
   git init
   ```

您将看到提示：`Initialized empty Git repository in ...`

步骤三：基本的文件操作

1. **创建一个新文件** `README.md`：

```
echo "# Git 实验项目" > README.md
```

2. **查看仓库状态**：

```
git status
```

您会看到 `README.md` 被列为未跟踪文件（Untracked files）。

3. **将文件添加到暂存区**：

```
git add README.md
```

4. **提交到本地仓库**：

```
git commit -m "初始化项目，添加 README.md 文件"
```

5. **查看提交历史**：

```
git log
```

6. **修改文件并查看差异**：

- 编辑 `README.md`，添加一行内容。

```
## 项目简介

这是一个用于练习 Git 基本操作的实验项目。
```

- 查看差异：

```
git diff
```

7. **将修改添加并提交**：

```
git add README.md
git commit -m "更新 README.md，添加项目简介"
```

步骤四：分支与合并

1. **创建并切换到新分支 `feature`**：

```
git switch -c feature
```

2. **在新分支中修改文件**：

   - 编辑 `README.md`，添加以下内容：

```
## 功能列表

- 学习 Git 的基本命令
- 掌握分支的使用
```

   - 保存并关闭文件。

3. **添加并提交更改**：

```
git add README.md
git commit -m "在 feature 分支中添加功能列表"
```

4. **切换回主分支**：

```
git switch master
```

5. **合并 `feature` 分支到主分支**：

```
git merge feature
```

   - 如果发生冲突，Git 会提示冲突的文件，需要手动解决。

6. **查看合并后的文件**：

   确认 `README.md` 包含 `feature` 分支中的更改。

7. **删除 `feature` 分支（可选）**：

```
git branch -d feature
```

步骤五：远程仓库交互

1. **在 GitHub 上创建一个新的空仓库**：
   - 登录 GitHub，点击右上角的 `+`，选择 `New repository`。
   - 输入仓库名称（如 `git-lab`），点击 `Create repository`。
2. **添加远程仓库地址**：

```
git remote add origin https://github.com/your_username/git-lab.git
```

   - 将 `your_username` 替换为您的 GitHub 用户名。
3. **推送本地仓库到远程仓库**：

```
git push -u origin master
```

   - 首次推送需要使用 `-u` 参数，设置默认的上游分支。
   - 您可能需要输入 GitHub 的用户名和密码（或使用令牌）。
4. **在 GitHub 上查看仓库**：

   刷新仓库页面，确认文件和提交记录已上传。
5. **从远程仓库拉取更新（模拟他人修改）**：
   - 在 GitHub 上直接编辑 `README.md` 文件，添加一行内容，然后提交。
   - 回到本地终端，执行：

```
git pull origin master
```

   - 查看 `README.md`，确认拉取的更改已更新到本地。

实验总结

通过本次实验，您完成了以下任务：

- **安装和配置 Git**：掌握了 Git 的安装和用户信息配置。
- **初始化本地 Git 仓库并进行基本操作**：理解了工作区、暂存区和本地仓库之间的关系，学会了使用 `git add` 和 `git commit`。

- **创建并合并分支**：学会了如何使用 `git branch`、`git switch` 和 `git merge` 来管理分支，理解了分支在开发中的重要性。
- **与远程仓库交互**：掌握了添加远程仓库 `git remote`、推送 `git push`、拉取 `git pull` 和查看远程仓库的方法。
- **查看并理解 Git 的状态和日志信息**：学会了使用 `git status`、`git log`、`git diff` 等命令。

通过这些操作，您已经完成了 Git 基本使用的核心流程，为以后的团队协作和代码管理打下了坚实的基础。

提示：

- **多实践**：Git 的命令很多，记住常用的即可，更多的是通过实践来熟悉。
- **遇到问题不要慌张**：如果在某个步骤遇到了错误信息，可以尝试阅读错误提示，或者搜索解决方案。
- **学习资源**：可以参考 Git 官方文档 或其他在线教程，获取更深入的了解。
- **版本控制的最佳实践**：
  - **及时commit**：保持小而频繁的提交，有助于追踪问题。
  - **写好commit信息**：清晰的提交信息可以让团队成员更容易理解更改内容。
  - **使用分支**：在分支上开发新功能，避免直接在主分支上进行实验性更改。