引言

在我们的日常生活中，无论是写作、绘画还是制作手工艺品，我们都有过这样的经历：创作过程中不断修改、完善，却又不小心覆盖了之前的版本，导致一些精彩的想法或创意丢失。在软件开发领域，这样的困扰同样存在。幸运的是，git作为一款分布式版本控制系统，为我们提供了完美的解决方案。

想象一下，你正在为一个重要的项目编写代码，过程中不断尝试新的思路和方法。通过git，你可以轻松地保存每个阶段的代码版本，就像是在时间线上标记了每一个重要的节点。当你想回到某个特定版本的代码时，只需简单操作，即可瞬间穿越回那个时刻。与此同时，git还支持多人协作，你和团队成员可以各自在本地进行开发，再将修改合并到一起，实现无缝衔接。

分布式版本控制系统（如git）与中央式版本控制系统（如SVN）

分布式版本控制系统（如git）与中央式版本控制系统（如SVN）在团队协作、代码管理和灵活性方面存在显著的差异。以下通过一个具体的例子来进一步说明这两者的区别。

假设有一个五人组成的开发团队，他们正在合作开发一个复杂的软件项目。在使用中央式版本控制系统（如SVN）时，团队成员会首先从中央仓库下载初始的代码。然后，每个人在自己的开发环境中进行代码的修改和添加新功能。每当完成一部分工作后，他们需要将新的代码提交到中央仓库。其他人如果需要这些新的代码，就需要从中央仓库同步到自己的本地环境。在这个过程中，如果多人同时修改了同一部分代码，就可能会出现代码冲突，需要手动解决。

然而，在使用分布式版本控制系统（如git）时，情况会有所不同。每个团队成员的本地机器上都有一个完整的代码库副本，即本地仓库。这意味着他们可以在没有网络连接的情况下进行代码的提交、查看历史记录和创建分支等操作。每个人都可以在自己的本地仓库中独立工作，而不会影响到其他人。当完成一部分工作后，他们可以将修改推送到中央仓库，或者从中央仓库拉取其他人的修改并合并到自己的本地仓库中。由于每个人的本地仓库都是完整的，因此冲突可以在本地解决，然后再推送到中央仓库，大大减少了冲突解决的复杂性。

此外，分布式版本控制系统的另一个优势是灵活性。由于每个团队成员都有自己的本地仓库，他们可以更容易地创建和管理分支。例如，一个开发者可以创建一个新的功能分支来尝试新的功能，而不会影响主线的开发。当功能开发完成后，他可以将这个分支合并回主线，或者选择将这个分支推送给其他团队成员进行审查。这种灵活性使得团队协作更加高效，也更容易追踪和管理代码的变更历史。

git介绍

git是一个开源的分布式版本控制系统，用于追踪代码或其他文件的变更历史。它最初由林纳斯·托瓦尔兹为辅助Linux内核开发而创建，现已成为最流行的版本控制系统之一。git不仅适用于大型软件开发项目，也适用于小型个人项目，其灵活性和强大的功能深受开发者爱。

git的核心优势在于其分布式特性。每个开发者都可以在自己的本地机器上拥有一个完整的代码库副本，这意味着开发者可以随时随地进行代码提交、查看历史记录以及创建分支等操作，无需依赖于中央服务器。这种分布式架构不仅提高了开发的灵活性和效率，还增强了项目的可靠性，因为即使中央服务器发生故障，开发者仍然可以在本地继续工作。

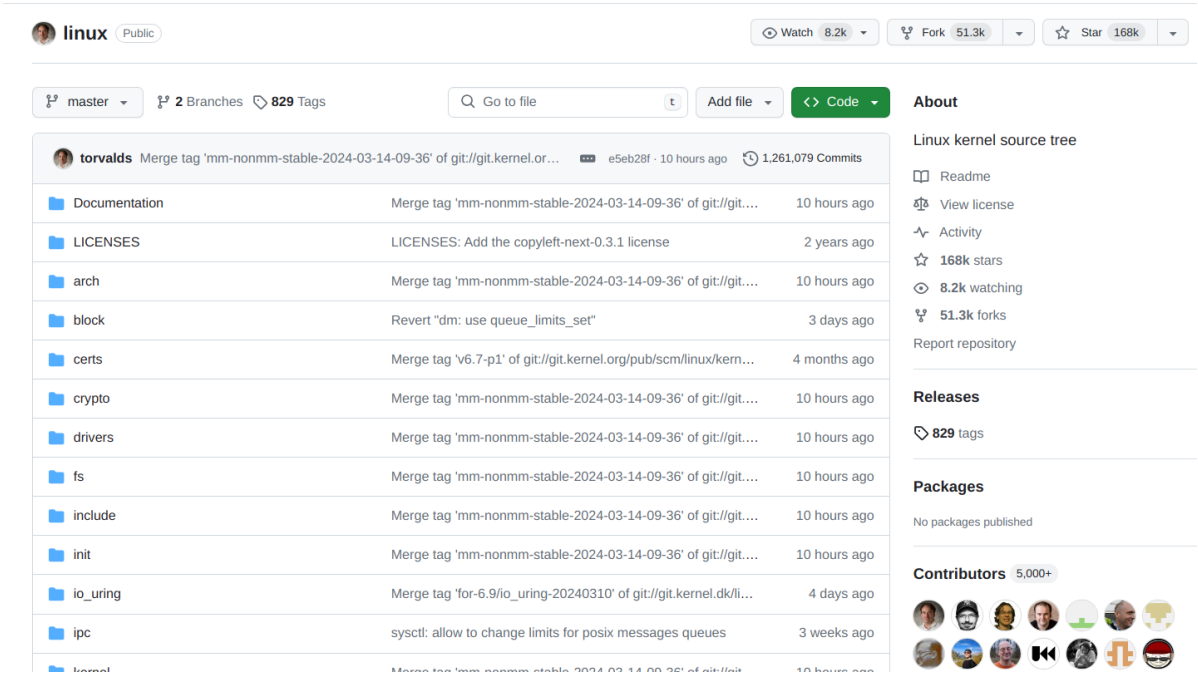
git的另一个显著特点是其强大的分支和合并功能。分支在git中非常轻量级，开发者可以轻松创建新的分支来尝试新的功能或修复错误，而不会干扰到主线的开发。当新功能或错误修复完成后，开发者可以将这些分支合并回主线，从而确保代码库的整洁和一致性。这种分支和合并的工作流程使得团队协作更加高效，也更容易管理和追踪代码的变更历史。

git、github、gitee的区别

git和gitee、gitHub其实是两个东西。gitHub、gitee是基于git 的代码托管服务网站，而git是一个版本的控制系统。简单来讲，git是一个帮我们管理代码版本的工具，而gitHub、gitee可以将git管理下的代码保存到云端，类似于我们的云盘。所以我们实际操作的是git这个工具。

而gitHub和gitee则是基于git实现的代码托管平台，它们提供了代码仓库托管、版本控制、分支管理、问题跟踪和协作等功能。这两个平台的主要区别在于其运营主体和定位。gitHub是全球最大的代码托管平台和开发者社区，由gitHub公司运营，面向全球开发者提供服务。而gitee则是中国的代码托管平台，主要面向中国开发者和团队，提供了更好的网络连接和中文界面。gitee还提供了企业版，提供私有仓库、团队协作和代码审查等功能。

简而言之，git是一个工具，用于在本地管理代码版本；而gitHub和gitee则是代码托管平台，用于在线托管和协作开发代码。开发者可以在本地使用git进行版本控制，然后将代码推送到gitHub或gitee等代码托管平台，与其他开发者进行协作和交流。



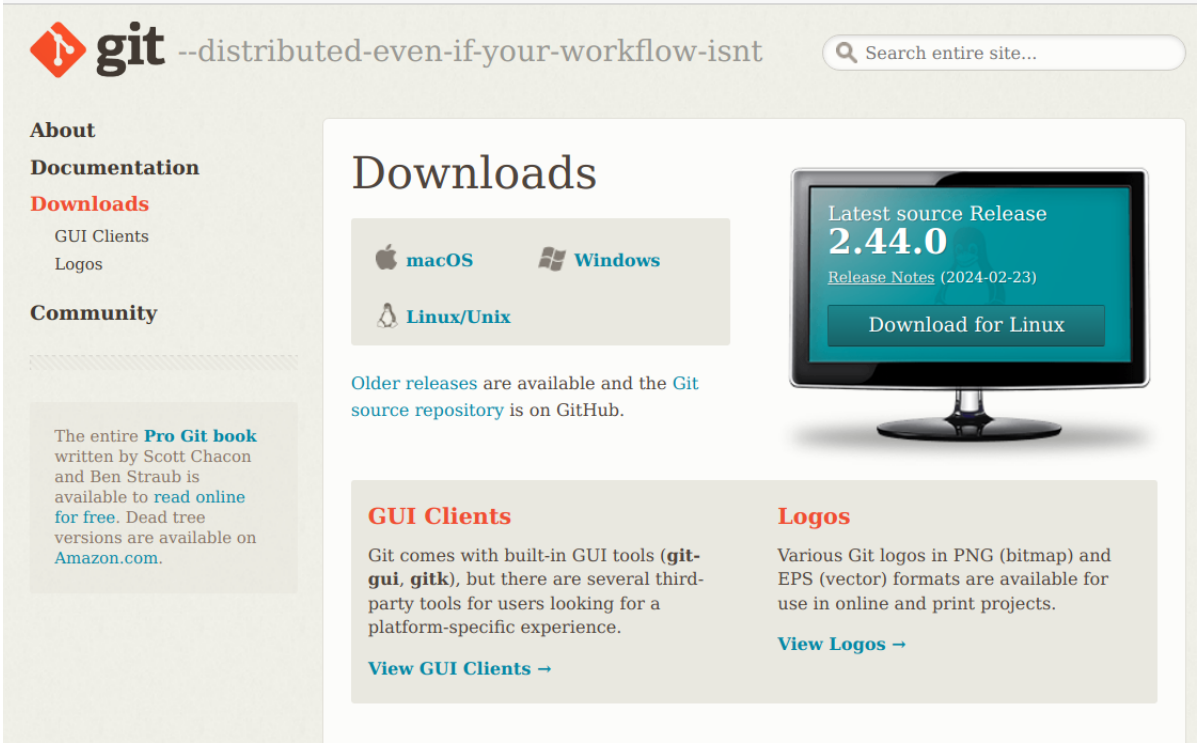
如图，这是gitHub上面一个开源项目的页面，可以看到创作者，stars数量，贡献者等。

下载安装

git在Windows系统下的安装教程

一、下载git安装包

1. 打开浏览器，访问git的官方网站：https://git-scm.com/downloads。
2. 在页面上选择“Windows”系统，然后点击“Download”按钮下载git安装包。



二、安装git

1. 下载完成后，找到下载的git安装包，双击运行。
2. 在弹出的安装向导中，点击“Next”开始安装。
3. 阅读许可协议，如果同意，请勾选“I agree to the git License Agreement”并点击“Next”。
4. 选择安装组件。通常情况下，建议选择默认设置，即安装所有组件。点击“Next”继续。
5. 选择安装位置。你可以自定义git的安装目录，或者保持默认设置。点击“Next”继续。
6. 选择开始菜单文件夹。同样，你可以自定义开始菜单中的文件夹名称，或者保持默认设置。点击“Next”继续。
7. 选择是否创建桌面快捷方式。根据个人喜好选择是否勾选“Create Desktop Shortcut”选项，然后点击“Next”。

8. 选择是否配置命令行终端。如果你希望在Windows的命令提示符或PowerShell中使用git，请勾选“Use the native Windows Secure Shell (instead of OpenSSH)”选项。点击“Next”继续。
9. 点击“Install”开始安装git。
10. 安装过程中，请耐心等待。安装完成后，点击“Finish”完成安装。

三、配置git

1. 安装完成后，打开Windows的命令提示符（CMD）或PowerShell。
2. 输入以下命令并回车，配置git的用户名和邮箱：

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

将`Your Name`替换为你的姓名，将`your_email@example.com`替换为你的邮箱地址。

3. 可以通过以下命令检查配置是否成功：

```
git config --global --list
```

如果看到之前设置的用户名和邮箱，说明配置成功。

四、验证git安装

1. 在命令提示符或PowerShell中输入以下命令：

```
git --version
```

如果成功显示git的版本号，说明git已经成功安装并配置好了。

基础使用（本地仓库）

一、初始化仓库

在命令行中，首先导航到你希望创建git仓库的目录。例如，假设我们有一个名为`my_project`的目录，你可以使用`cd`命令进入该目录：

```
cd path/to/my_project
```

接下来，使用`git init`命令初始化git仓库：

```
git init
```

此命令会在`my_project`目录下创建一个名为`.git`的子目录，这个子目录包含了git仓库的所有元数据对象。

二、文件的添加与提交

在git中，文件的生命周期包括三个状态：已修改（modified）、已暂存（staged）和已提交（committed）。

1. **添加文件到暂存区**

使用`git add`命令可以将文件添加到暂存区。例如，假设你创建了一个新文件`example.txt`，你可以使用以下命令将其添加到暂存区：

```
git add example.txt
```

如果你想添加当前目录下的所有文件，可以使用`.`作为参数：

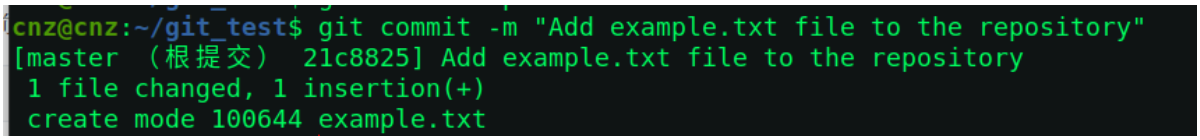
```
git add .
```

2. 提交更改

当你对文件做了修改并添加到暂存区后，可以使用`git commit`命令提交这些更改。提交时，最好附上一条有意义的提交信息，描述你所做的更改：

```
git commit -m "Add example.txt file to the repository"
```

`-m`参数后面是你为这次提交所添加的描述信息。



三、查看仓库状态与提交历史

1. 查看仓库状态

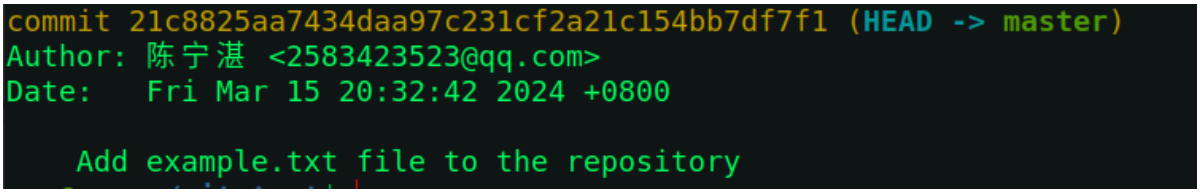
使用`git status`命令可以查看当前仓库的状态，包括哪些文件已经被修改、哪些文件已经被添加到暂存区等：

```
git status
```

2. 查看提交历史

`git log`命令可以用来查看仓库的提交历史：

```
git log
```



你可以通过添加参数来定制输出的格式，例如使用`--oneline`参数以更简洁的方式显示提交历史：

```
git log --oneline
```

四、撤销更改与重置

1. 撤销工作区的修改

如果你修改了文件但还没有添加到暂存区，并希望撤销这些修改，可以使用`git checkout`命令：

```
git checkout -- example.txt
```

这会将`example.txt`文件恢复到最后一次提交的状态。

2. **撤销暂存区的修改**

如果你已经将文件添加到暂存区，但还没有提交，并希望撤销添加到暂存区的操作，可以先使用`git reset`命令将文件从暂存区移除，然后再使用`git checkout`命令撤销工作区的修改：

```
git reset HEAD example.txt
git checkout -- example.txt
```

3. **撤销提交**

如果你想撤销某次提交，可以使用`git reset`命令。例如，要撤销最近一次的提交，可以使用：

```
git reset --hard HEAD~1
```

请注意，`--hard`选项会丢弃最近一次提交的所有更改。这是一个强力的操作，使用前请确保你真的想要这么做，并且已经备份了重要的数据。

本地仓库关联远程仓库

一、前提准备

在开始之前，请确保你的计算机上已经安装了git，并且你已经在本地创建了一个git仓库（本地仓库）。如果你还没有本地仓库，请参考前面的教程创建一个。

注册gitee账号(自行按提示注册):

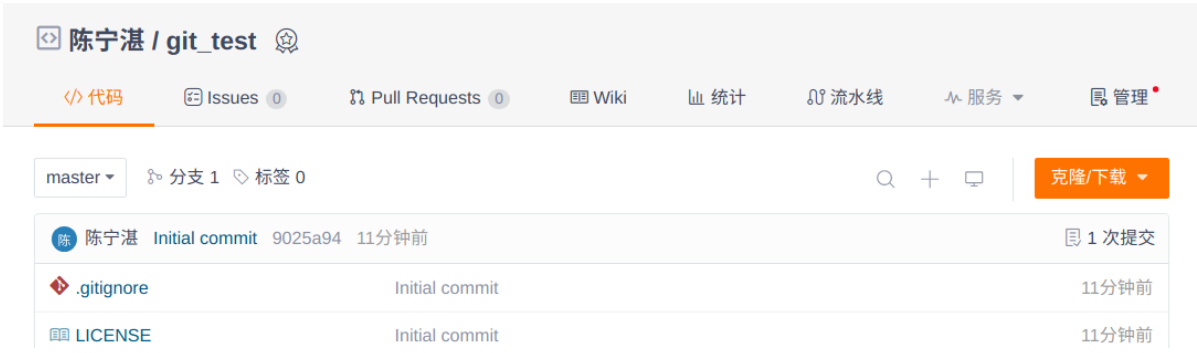
二、将本地仓库关联到远程仓库

1. 在代码托管平台上创建远程仓库

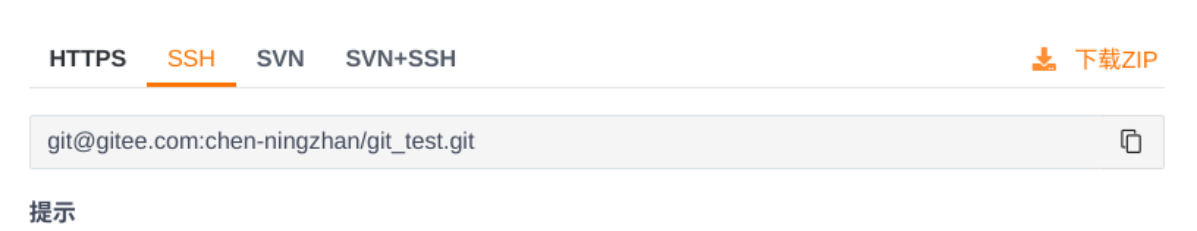
首先，登录到你的代码托管平台账户，并创建一个新的远程仓库

2. **在本地仓库中设置远程仓库的URL**

点击右上角的“克隆/下载”



复制ssh（这里需要配置SSH key，可以自行csdn或者其他资料解决）



打开命令行界面，并导航到你的本地仓库所在的目录。然后，使用`git remote`命令来设置远程仓库的URL。通常，我们将远程仓库命名为`origin`，但你也可以使用其他名称。

```
git remote add origin <远程仓库的URL>
```

将`<远程仓库的URL>`替换为你在代码托管平台上获得的URL。例如：

```
git remote add origin https://github.com/username/repo.git
```

3. 验证远程仓库是否设置成功

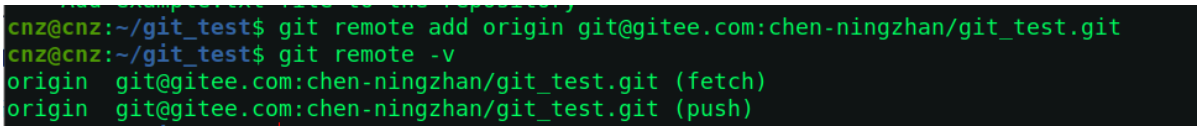
使用`git remote -v`命令来验证远程仓库是否已成功设置，并显示其URL：

```
git remote -v
```

如果一切正常，你应该看到类似于以下的输出：

```
origin  https://github.com/username/repo.git (fetch)
origin  https://github.com/username/repo.git (push)
```

这表明你的本地仓库已经与名为`origin`的远程仓库关联起来，并且你可以从这个远程仓库拉取（fetch）和推送（push）更改。



一、准备本地文件夹

首先，确保你有一个本地的文件夹，这个文件夹包含了你想要进行版本控制的项目文件。

二、初始化本地git仓库

在本地文件夹中打开命令行（终端），执行以下命令来初始化一个新的git仓库：

```
git init
```

这个命令会在当前文件夹中创建一个名为`.git`的子文件夹，用于存储版本控制的所有信息。

三、将文件添加到暂存区

使用`git add`命令将你的项目文件添加到git的暂存区。如果你想添加当前文件夹下的所有文件和子文件夹，可以执行：

```
git add .
```

这里的`.`代表当前目录下的所有文件和子目录。如果你只想添加特定的文件或文件夹，可以替换`.`为文件或文件夹的路径。

四、提交暂存区的更改

接下来，使用`git commit`命令将暂存区的内容提交到本地仓库，并附加一条提交信息：

```
git commit -m "Initial commit with project files"
```

这里的`"Initial commit with project files"`是提交信息，你可以根据实际的更改内容替换成其他描述性的文字。

五、关联远程仓库（远程仓库为空）

如果远程仓库是空的或者你刚刚创建了一个新的远程仓库，你需要将本地仓库与远程仓库关联起来，并推送本地分支到远程仓库。

1. 添加远程仓库地址

首先，使用`git remote add`命令添加远程仓库的地址：

```
git remote add origin <远程仓库地址>
```

将`<远程仓库地址>`替换为你的远程仓库的URL，这通常是一个以`.git`结尾的URL，比如从gitHub、gitLab等平台获得的地址。

2. 推送本地分支到远程仓库

接下来，使用`git push`命令将本地分支推送到远程仓库：

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

这里假设你的默认分支是`master`。如果你的默认分支名不是`master`，请替换为实际的分支名。`-u`参数表示设置上游（upstream）仓库，以后你可以直接使用`git pull`或`git push`而不需要指定远程仓库和分支名。

六、关联远程仓库（远程仓库已有文件）

如果远程仓库已经存在文件，你需要先从远程仓库拉取（fetch）最新的代码，并将本地代码与远程代码合并或变基，然后再推送（push）到远程仓库。

1. 拉取远程仓库的最新代码

```
git fetch origin
```

这会将远程仓库的最新代码下载到本地，但不会自动合并或修改你的工作。

2. 合并或变基远程代码

如果远程仓库的代码和你的本地代码有冲突，你需要先解决这些冲突。你可以通过合并（merge）或变基（rebase）来整合代码。

合并代码：

```
git merge origin/master
```

变基代码（推荐在没有推送本地更改到公共仓库时使用）：

```
git rebase origin/master
```

如果合并或变基过程中产生冲突，你需要手动解决这些冲突，然后提交解决后的更改。

3. 推送本地分支到远程仓库

解决完所有冲突并提交后，你可以将本地分支推送到远程仓库：

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

同样，这里假设你的默认分支是`master`。如果你的默认分支名不是`master`，请替换为实际的分支名。

七、验证连接

最后，你可以使用以下命令来检查你的本地仓库是否已成功与远程仓库连接：

```
git remote -v
```

这个命令会列出所有远程仓库的信息，包括仓库的名称、远程地址以及对应的引用（通常是分支）。你应该能在输出中看到你刚刚添加的远程仓库（通常名为`origin`）的信息。