第二次提交

2024.01.22 bug清单

为什么要学习git，非程序员完全可以使用，版本管理太方便了，方便我有条理的管理所有项目

## 问题一2222master

#### 本门课程，打算如何介绍git

答: 我了解到老师打算从五个方面介绍git，首先介绍git是什么，它是一个分布式版本控制工具，通过和集中式版本控制工具做对比，其次交我们如何安装git，然后介绍git的常用命令，接着是重点，解释git的分支，最后解释idea使用git

## 问题二、

#### 本门课程，打算如何解释github

答:老师打算从六个方法介绍。首先解释如何在github创建远程库，其次把代码推送到远程库，然后把从远程库拉取同事上传的代码，接着教如何clone别人发布在github的项目代码。然后是如何通过ssh不用账号，最后是解释idea使用github。

## 问题三、

#### 本门课程，打算如何解释gitee

答:因为gitee是模仿github，所以老师打算从三方面简单讲解，首先是gitee建立远程库，然后解释idea使用github。最后是重点，连接github进行代码复制和迁移。

## 问题四、

#### 本门课程，打算如何解释gitlab

答:两方面，首先学习gitlab服务器的搭建和部署。其次学习idea使用gitlab

## 问题五、

#### 首先介绍git是什么。然后它是一个分布式版本控制工具，请和集中式版本控制工具做对比。

答:Git是一个开源的分布式版本控制系统，用于追踪文件的改变、协调多人在同一个代码仓库中的开发以及管理代码的版本。它最初由Linux创始人Linus Torvalds于2005年创建，是目前最流行的版本控制工具之一。

相较于集中式版本控制系统（例如SVN），Git采用了分布式的模式，每个开发者都可以在本地拷贝完整的代码仓库。这意味着开发者可以在本地提交代码和进行版本控制的操作，而不需要依赖网络连接。当需要进行代码同步或者与其他开发者协作时，可以通过推送（push）和拉取（pull）的方式与远程代码库进行交互，将自己的修改与他人的修改进行合并。

与集中式版本控制工具相比，Git具有以下优势：

分布式：每个开发者都有一份完整的代码仓库，具备更好的代码管理和版本控制能力。

本地化操作：它不需要经常与远程代码库交互，而是在本地进行提交、分支操作等。

易于分支管理：Git的分支功能非常强大，可以轻松创建、合并和删除分支，支持并行开发和灵活的代码流程。

效率和速度：Git的设计使其能够快速处理大型项目和大量的历史记录。

总的来说，Git相较于集中式版本控制系统具有更强大的分支管理能力、更高效的代码管理和更好的版本控制能力，因此在开发中被广泛使用。

## 问题六

#### 简述，git的add，commit命令和远程库github,gitee,gitlab之间的关系

答:在Git中，使用 add 命令将工作区的修改添加到暂存区， commit 命令将暂存区的修改提交到本地代码仓库。

在与远程库（如GitHub、Gitee、GitLab）进行协作时，可以将本地仓库同步到远程库，使其他开发者可以查看、下载和贡献代码。这里是一种常见的流程：

在本地使用 add 命令将修改添加到暂存区。

使用 commit 命令将暂存区的修改提交到本地仓库。

将本地仓库推送（push）到远程库。

远程库（如GitHub、Gitee、GitLab）是一个托管代码的服务器，它们提供了协同开发所需的功能，例如版本控制、代码浏览、合并请求等。开发者可以通过将本地仓库推送到远程库，将自己的修改与其他开发者共享。

GitHub、Gitee、GitLab都是流行的远程代码托管平台，它们基于Git版本控制系统，提供了类似的功能。每个平台都有自己的特点和优势，如GitHub针对开源项目非常受欢迎，Gitee在中国用户中较为流行，GitLab则提供了更丰富的CI/CD功能。开发者可以根据自己的需求和偏好选择使用其中之一（或同时使用多个）来协作开发和管理代码。

总结起来， add 和 commit 命令是Git提供的用于将修改保存到本地代码仓库的操作，而GitHub、Gitee、GitLab是远程代码托管平台，用于托管和协同开发代码。开发者可以使用Git的命令将修改提交到本地仓库，并通过推送操作将本地仓库同步到远程库，与其他开发者协作。

## 问题7

#### git设置用户签名的命令和作用

**答:用户签名用于标识代码的提交者。当你使用 commit 命令提交代码时，Git会将你设置的用户名和邮箱信息嵌入到提交记录中，记录下是谁提交了这部分代码。**

在Git中，可以使用以下命令来设置用户签名：

**设置全局用户签名：**

**git config --global user.name "Your Name"**

**git config --global user.email "your.email@example.com"**

**这些命令设置了默认的用户名称和电子邮件地址，会应用于所有的Git仓库。**

**设置当前仓库特定的用户签名：**

**git config user.name "Your Name"**

**git config user.email "your.email@example.com"**

**这些命令设置了当前仓库的用户名称和电子邮件地址。**

设置用户签名有助于为代码贡献者建立身份和信任，并提供了一种简单的方式来追溯和识别提交者。这对于团队协作和开源项目非常重要。

问题八

git解释初始化本地库，以及查看本地库的状态，说一下不同情况下的效果。

答:初始化本地库是指在本地计算机上创建一个新的Git仓库，以开始进行版本控制。在初始化之后，Git会为该仓库创建一些必要的文件和目录，以跟踪和管理项目的版本历史信息。

在命令行中，可以使用以下命令来初始化一个Git仓库：

git init

执行此命令后，Git会在当前目录下创建一个新的空白仓库，该仓库的文件和目录将被用于存储项目的版本历史记录。

在不同情况下，初始化本地库的效果可能会有所不同：

在一个空的目录中执行 git init 命令：此时Git会在当前目录下创建一个新的空白仓库，并为该仓库创建一个 .git 子目录。该目录包含了Git仓库的所有必要文件和目录。

在一个已有项目的目录中执行 git init 命令：此时Git会将该目录变成一个Git仓库，即把项目的当前状态纳入版本控制。在该目录下的所有文件和目录将被追踪，并且Git会开始记录所有的修改。

查看本地库的状态可以使用 git status 命令。执行该命令时，Git会列出当前工作目录中的文件，并显示文件的状态信息。

在一个已初始化的本地库中， git status 命令的输出内容可能会有以下几种情况：

如果没有进行任何修改，输出类似于：

On branch master

nothing to commit, working tree clean

(主线分支，无事可做，工作树干净)

这表示当前工作目录中没有未提交的修改。

如果有修改但未暂存，输出类似于：

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

(主线分支，有一些改变，还没有提交。使用git add 命令添加到暂存区)

modified: <file1>

modified: <file2>

no changes added to commit (use "git add" and/or "git commit -a")

这表示有一些已修改的文件，但是还没有将它们暂存。

如果有修改并且已经暂存，输出类似于：

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: <file1>

modified: <file2>

这表示有一些已修改的文件，并且这些修改已经被暂存，可以将它们提交。

如果有新添加的文件，输出类似于：

On branch master

Untracked files:(未追踪的文件，使用git add 文件添加到暂存区)

(use "git add <file>..." to include in what will be committed)

<file3>

<file4>

这表示有一些新添加的文件，但是它们还没有被跟踪。

综上所述，不同情况下的初始化本地库和查看本地库状态的效果可能会根据项目的不同而有所差异，但是可以通过使用Git命令来了解和操作本地库的状态。

问题九

解释on branch master

答:"on branch master" 是指当前所在的Git分支是"master"分支。

问题十

使用vim命令，创建一个文本，写入一段内容，保存

答:若想使用Vim命令在命令行中创建文本、写入内容并保存，可以按照以下步骤进行：

打开命令行终端。

使用  cd  命令导航到你想要创建文本文件的目录。

cd Desktop

使用Vim命令创建并编辑文本文件。运行以下命令来创建一个名为 example.txt 的文本文件，并用Vim编辑器打开它：

vim example.txt

进入Vim编辑器后，在命令行底部会显示一个 : 符号。你可以在此处输入各种Vim命令。

若要切换到插入模式以输入文本，请按下 i 键。你将看到底部显示"-- INSERT --"。

输入你要写入到文件中的文本。

按下 Esc 键以退出插入模式。

输入 :wq 命令，然后按下回车键保存文件并退出Vim编辑器。

 w 代表写入（保存）文件， q 代表退出Vim。

现在你应该已经创建了一个名为 example.txt 的文本文件，并将内容保存在其中。

问题十一

命令行vim编辑文件，复制粘贴的命令是什么

答:复制文本：

进入Vim编辑器后，将光标移动到要复制的起始位置。

按下 v 进入可视模式（逐字符选择），或按下 V 进入可视行模式（逐行选择）。

移动光标以选择要复制的文本。

按下 y 复制选定的文本。

粘贴文本：

将光标移动到要粘贴的位置。

按下 p 在光标后粘贴文本，或按下 Shift + p 在光标前粘贴文本。

问题十二

命令行vim编辑文件，整行复制的命令是什么

答:在命令行中使用Vim编辑器时，复制整行的命令是  yy 。

移动光标至要复制的行。

输入  yy  命令以复制当前行。

光标将保持在原来的位置，同时整行被复制到了剪贴板中。

你可以通过将光标移动到其他位置，然后按下  p  命令将整行粘贴到光标下方，或使用  P  命令将整行粘贴到光标上方。

问题十三

git从暂存区删除文件的命令

答: 要将文件从 Git 暂存区中删除，可以使用以下命令：

git rm --cached <文件名>

该命令将从暂存区中删除指定的文件，但会保留在工作区中。你需要将  <文件名>  替换为实际要删除的文件名。

如果你想要删除多个文件，可以使用通配符  \*  来匹配多个文件，例如：

git rm --cached \*.txt

注意，执行  git rm --cached  命令后，文件将从 Git 的跟踪列表中消失，但实际文件还存在于工作目录中。如果你希望删除工作目录中的文件，需要使用  git rm  命令：

git rm <文件名>

执行该命令后，文件将从暂存区和工作区中同时删除。

问题十四

git如何将文件提交到github

答: 要将 Git 暂存区中的文件提交到 GitHub，需要以下步骤：

你可以使用  git add  命令将修改的文件添加到暂存区，例如：

git add <文件名>

使用  git commit  命令提交暂存区的文件。在提交时，需要为本次提交提供一个有意义的提交消息，描述你所做的修改内容，例如：

git commit -m "提交消息"

在 GitHub 上创建一个远程代码仓库（如果尚未创建）。在 GitHub 上点击 "New" 按钮创建一个新的仓库，并按照提示填写相关信息。

将本地代码仓库与远程代码仓库关联起来。使用以下命令添加远程代码仓库的地址：

git remote add origin <远程仓库地址>

其中， <远程仓库地址>  是你所创建的 GitHub 仓库的 URL。

将本地代码推送到远程代码仓库。使用以下命令将本地代码推送到远程仓库：

git push -u origin <分支名>

其中， <分支名>  是要推送的分支名，默认情况下为  master 。

如果一切顺利，Git 将把你的本地代码推送到 GitHub 的远程代码仓库中。你可以在 GitHub 上的仓库页面上查看提交记录和修改内容。

问题十五

git commit -m "提交信息" 后面不应该加文件名吗？

在使用git commit命令时，通常需要指定要提交的文件名或文件路径。

如果没有指定文件名，Git 会将所有已暂存的文件一并提交。

问题十六

介绍git reflog命令

答: git reflog  命令用于显示 Git 仓库中的引用日志。它记录了本地仓库中的各种引用（分支、HEAD、标签等）的更新历史，包括分支切换、提交、合并等操作。 git reflog  可以帮助你查看并恢复误操作、丢失的提交或分支。

例如：

1f54c1e (HEAD -> master) HEAD@{0}: commit: Fix bug #123

8d7a543 HEAD@{1}: commit: Add new feature

4a7b9bf HEAD@{2}: checkout: moving from feature-branch to master

69e7c6c HEAD@{3}: commit: Update README file

其中，每条记录都包含一个唯一的引用指针（如  HEAD@{0} ）、引用所指向的提交的 SHA-1 值、引用的名称（如  master 、 feature-branch ）、操作（如  checkout 、 commit ）和操作的说明。

通过查看  git reflog ，你可以找到被删除的分支、丢失的提交，或者回滚到之前的某个状态。可以使用  git checkout  或  git restore  命令结合引用日志中的提交 SHA-1 值来恢复到指定的历史状态。

问题十七

具体解释以下日志。

 1f54c1e (HEAD -> master) HEAD@{0}: commit: Fix bug #123

 1f54c1e  是一个提交的 SHA-1 值，表示最近的提交。

 (HEAD -> master)  表示当前所在的分支是  master  分支。

 HEAD@{0}  表示这是最近的一次操作。

 commit: Fix bug #123  是这次提交的说明，说明了这次提交解决了一个名为 "bug #123" 的问题。

 8d7a543 HEAD@{1}: commit: Add new feature

 8d7a543  是另一个提交的 SHA-1 值，前一个操作的提交。

 HEAD@{1}  表示这是倒数第二次的操作。

 commit: Add new feature  是这次提交的说明，意味着这次提交添加了一个新功能。

 4a7b9bf HEAD@{2}: checkout: moving from feature-branch to master

 4a7b9bf  是另一个提交的 SHA-1 值，前两个操作的提交。

 HEAD@{2}  表示这是倒数第三次的操作。

 checkout: moving from feature-branch to master  表示这次操作是通过从  feature-branch  分支切换到  master  分支来完成的。

 69e7c6c HEAD@{3}: commit: Update README file

 69e7c6c  是另一个提交的 SHA-1 值，前三个操作的提交。

 HEAD@{3}  表示这是倒数第四次的操作。

 commit: Update README file  是这次提交的说明，说明了这次提交更新了 README 文件。

通过分析这些日志，你可以了解到仓库中的每个操作并且可以根据需要恢复到特定的历史状态。

问题十八

是不是每commit一次就会有一条这样的日志？

答:是的，每次提交代码，Git 会自动生成一条类似的日志记录。这些日志记录被保存在 Git 的日志中，用于跟踪项目的修改历史和操作记录。每个提交都有一个唯一的 SHA-1 值，以及包含提交说明、作者、时间戳等其他元数据。这些日志记录可以通过使用 Git 命令行工具或 Git 图形界面工具来查看和管理。这些日志对于项目的版本控制和协作非常重要，能够追踪和还原项目的各个开发阶段和改动。

问题十九

用git reset做版本穿梭

答:是的，使用 git reset 命令也可以在Git中进行版本穿梭。

要回退到特定的提交，可以使用 git reset 命令，后面跟上要回退到的提交的引用。有三种重置模式可供选择：

软重置（soft reset）：通过 git reset --soft <commit> 命令，将HEAD指针和当前分支指向指定的提交，但不会修改工作目录和暂存区。这意味着你可以重新提交上一个提交，或者继续对其进行修改和暂存。

混合重置（mixed reset）：通过 git reset --mixed <commit> 命令（该命令是默认的），将HEAD指针和当前分支指向指定的提交，并重置暂存区，但不会修改工作目录。这意味着你可以重新选择要提交的内容。

硬重置（hard reset）：通过 git reset --hard <commit> 命令，将HEAD指针和当前分支指向指定的提交，并重置暂存区和工作目录到指定的提交。这意味着你会丢失指定提交以后的所有修改。

例如，如果你要重置到提交4a7b9bf并丢弃之后的所有更改，可以使用以下命令：

git reset --hard 4a7b9bf

问题二十

什么叫丢失之后的所有更改

答:使用版本迁移（如 git reset ）会影响未提交的代码和已提交的代码的恢复方式。

如果你有未提交的代码更改而进行版本迁移，如使用 git reset --hard <commit> ，那么所有未提交的更改将会被永久删除，且无法通过简单的操作来恢复。这是因为 --hard 选项会重置暂存区和工作目录，丢弃未提交的更改。

然而，对于已经提交的代码，你可以通过版本迁移来恢复它们。通过指定合适的提交，你可以重置分支（或HEAD）来回退到先前的提交。这种操作不会影响已经正式提交的代码和提交历史。

问题二十一

解释git的分支是什么

答:Git的分支是指将代码库中的一些特定提交版本（commit）的引用。分支可以看作是代码库中的一个独立的线路，它可以独立存在并进行修改、添加或移除提交。使用分支可以方便地实现并行开发和协作，各个分支之间相互独立，不会相互影响。

在Git中，通常会有一个默认的主分支（通常是 master 分支），它是最新和最稳定的代码版本。当需要进行新功能开发或修复bug时，可以基于主分支创建一个新的分支。在这个新的分支上进行修改，不会直接影响到主分支，因此可以安全地进行并行开发。一旦新功能开发或bug修复完成，可以将这个分支合并回主分支，将修改的内容应用到主分支上。

分支在Git中有很多应用场景，比如团队合作开发时，每个成员可以基于主分支创建自己的分支，在分支上进行开发，最后合并到主分支；也可以用于为不同版本或不同环境创建独立的分支，方便代码管理和部署。总之，分支是Git中非常重要的概念，它为代码的管理、版本控制和协作提供了很大的灵活性。

问题二十二

git创建分支，切换分支和查看分支的命令。

答:在Git中，可以使用以下命令来创建和查看分支：

创建分支：

 git branch <branch-name> ：创建一个新的分支，并且该分支的基线（起点）与当前所处的分支相同。

 git branch <branch-name> <commit> ：在指定提交（commit）上创建一个新的分支。

查看分支：

 git branch ：查看当前仓库中所有的分支，以及当前所处的分支（会在当前分支名字前面标记一个  \* ）。

 git branch -a ：查看包括远程分支在内的所有分支。

 git branch --merged ：查看已经与当前分支合并的分支。

 git branch --no-merged ：查看还未与当前分支合并的分支。

需要注意的是，创建分支只是在Git中保存一个指针指向某个提交，创建分支不会自动切换到新建分支上，如果想要切换到新建分支，可以使用  git checkout  命令。

例如，创建一个名为  feature  的新分支并切换到该分支上的命令：

git branch feature

git checkout feature

或者，创建一个名为  feature  的新分支并切换到该分支上的合并命令：

git checkout -b feature

问题二十三

用git branch -v和git branch有什么区别

答: git branch  和  git branch -v  命令的区别在于输出的详细程度。

 git branch  命令用于列出所有的分支，但只显示分支的名称。 例如，执行  git branch  命令可能得到以下输出：

\* master

develop

feature

hotfix

上述输出告诉我们当前所处的分支是master分支，并且存在develop、feature和hotfix等其他分支。

 git branch -v  命令在列出所有分支的同时，还会显示每个分支关联的最新提交（commit）的信息。 例如，执行  git branch -v  命令可能得到以下输出：

\* master 4a2d8c4 Latest commit on master branch

develop df9e3b1 Latest commit on develop branch

feature 789cdef Latest commit on feature branch

hotfix abcd123 Latest commit on hotfix branch

上述输出除了显示分支的名称外，还显示每个分支关联的最新提交的哈希值（例如4a2d8c4、df9e3b1等）和对应的提交消息。

因此， git branch -v  命令提供了更详细的分支信息，可以更好地了解每个分支的状态和最新的提交。

问题二十四

写一下git从master分支，切换到hotfix分支，修改文件后，然后add和commit到远程库的命令

答:下面是一系列在 git 中进行操作的命令，用于从 master 分支切换到 hotfix 分支、修改文件、将修改添加到暂存区和提交到远程仓库：

确保当前所在的分支是 master 分支：

git branch

如果当前分支不是 master，则先使用  git checkout master  命令切换到 master 分支。

切换到 hotfix 分支

git checkout hotfix

修改文件： 通过编辑相关文件进行修改。

将修改的文件添加到暂存区：

git add <file1> <file2> ...

或者使用以下命令将所有修改的文件都添加到暂存区：

git add .

提交修改到本地仓库：

git commit -m "Commit message"

其中，"Commit message" 是对该次提交的描述信息。

将本地的 hotfix 分支推送到远程仓库：

git push origin hotfix

这将会把 hotfix 分支的修改推送到远程仓库。

问题二十五

然后再把hotfix分支的代码合并到master

答:要将 hotfix 分支的代码合并到 master 分支，您可以按照以下步骤进行操作：

确保您当前所在的分支是 master 分支：

git branch

 如果当前分支不是 master，则使用  git checkout master  命令切换到 master 分支。

合并 hotfix 分支到 master 分支：

git merge hotfix

 这将会将 hotfix 分支的代码合并到 master 分支。

提交合并的修改到本地仓库：

git commit -m "Merge hotfix into master"

 请提供适当的提交信息描述合并操作。

将合并后的修改推送到远程仓库：

git push origin master

 这将会将 master 分支的修改推送到远程仓库。

问题二十六

模拟一下，master和hotfix分支，都对文件同一行进行修改，hotfix合并到master时，发生冲突，如何解决

答:当在master分支和hotfix分支上对同一文件的同一行进行修改时，如果在合并hotfix分支到master分支时发生冲突，可以按照以下步骤解决冲突：

确保你在本地的工作区是干净的，没有未提交的更改。使用 git status 命令检查当前状态。

在终端中，切换到master分支（假设你当前位于hotfix分支上）。使用 git checkout master 命令切换到master分支。

运行 git merge hotfix 命令将hotfix分支合并到master分支。此时可能会出现冲突。

Git会自动将冲突标记在文件中，你可以使用文本编辑器打开包含冲突的文件，看到类似下面的内容：

<<<<<<< HEAD (master)

此处是master分支的内容

=======

此处是hotfix分支的内容

>>>>>>> hotfix

在编辑器中手动解决冲突，你可以选择保留master分支的内容、保留hotfix分支的内容或合并两者的内容。修改后的文件保存。

解决冲突后，在终端运行 git add <冲突的文件> 命令将解决的文件标记为已解决。

运行 git commit 命令来提交合并的更改。

请注意，在解决冲突时，你需要根据实际情况修改文件并标记为已解决，然后提交更改。最后，可以使用 git log 命令检查合并结果。

问题二十七

git命令，如何实现跨公司合作写代码，举例华山派公司有远程库叫华山剑法，华山派公司领导令狐冲，邀请日月神教公司的东方不败，fork和clone华山剑法远程库，改进华山剑法代码，改好了pull给华山派公司老板岳不群审批，审批通过后合并到华山剑法远程库中。

答:下面是使用Git命令实现跨公司合作写代码的步骤，按照你提供的例子：

令狐冲邀请东方不败fork和clone华山剑法远程库：

东方不败在日月神教公司的Git平台上fork华山剑法远程库，创建自己的远程库副本；

git clone <华山剑法远程库URL>

东方不败在本地仓库中创建分支：

git checkout -b feature/东方不败改进

东方不败进行代码改进，并将改进的代码推送到自己的远程库：

git add .

git commit -m "改进华山剑法"

git push origin feature/东方不败改进

东方不败发起Pull请求，请求将改进的代码合并到华山剑法远程库：

在日月神教公司的Git平台上发起Pull请求，将东方不败改进的代码合并到华山剑法远程库；

请注意，在发起Pull请求前，确保你的代码已经推送到自己的远程库。

岳不群审批并合并Pull请求：

岳不群作为华山派公司的老板，在华山剑法远程库的页面上审批东方不败的Pull请求，并选择合并代码到主分支。

以上步骤可以帮助华山派公司的领导令狐冲和日月神教公司的东方不败实现跨公司合作写代码，并将改进的代码合并到华山剑法远程库中。请确保团队成员遵循合适的工作流程和代码管理规范执行这些操作。

问题二十八

如何在github上创建远程库，有什么注意点

答:在 GitHub 上创建远程库（即创建一个新的 empty repository）非常简单。以下是在 GitHub 网站上创建远程库的步骤：

打开 GitHub 网站 (https://github.com/) 并登录到您的账号。

在页面右上角，点击头像旁边的加号 “+” 按钮，然后选择 "New repository"。

在 "Repository name" 输入框中，输入您想要的远程库的名称。

可选：在 "Description" 输入框中，输入远程库的描述信息。

可选：选择远程库的 Synchronization（同步）选项。

可选：选择是否要添加一个初始的 README 文件。

可选：选择要添加到 .gitignore 文件中的模板，以忽略特定类型的文件。

可选：选择要添加到许可证文件中的许可证模板。

点击 "Create repository" 创建远程库。

注意事项：

在创建远程库时，确保为远程库选择一个容易理解的名称，并提供可用的描述信息。

如果选择了初始的 README 文件和 .gitignore 文件模板，GitHub 将会自动向您的仓库添加这些文件，并根据您选择的模板进行配置。

如果您希望其他人能够贡献代码到您的远程库，可以将其设置为公共库。如果只想保留自己对库的访问权限，可以将其设置为私有库（可能需要付费）。

在远程库中，您可以使用 "Settings" 选项卡来配置其他功能，例如许可证、协作访问权限等。

这些步骤应该能帮助您在 GitHub 上创建一个新的远程库。完成创建后，您可以通过  git clone  命令将该库克隆到本地，并在本地进行代码的编辑和提交。

问题二十九

git查看远程库的别名，和创建别名

在 Git 中，远程库的别名被称为 "remote"。要查看远程库的别名，您可以使用以下命令： git remote

该命令将列出您当前配置的所有远程库的别名。如果您的 Git 仓库没有配置任何远程库，该命令将没有任何输出。

要创建一个远程库的别名，您可以使用以下命令：

git remote add <别名> <远程库的 URL>

其中， <别名>  是您想要为远程库设置的别名， <远程库的 URL>  是远程库的地址。

例如，如果您想要为一个名为 "origin" 的远程库设置一个别名，并将其 URL 设置为 "https://github.com/example/example-repo.git"，可以使用以下命令：

git remote add origin https://github.com/example/example-repo.git

创建别名后，您可以使用别名来引用远程库，例如使用  git push 、 git fetch 、 git pull  等命令。

注意：在使用别名时，请确保别名是唯一且易于理解的，这样可以更方便地管理远程库。

问题三十

git创建别名后，查看远程库别名输出什么

答:在 Git 中，创建了远程库的别名后，使用以下命令  git remote  查看远程库别名，将输出您当前配置的所有远程库的别名。

 git remote -v  命令，它会显示远程库的别名及其对应的 URL。

$ git remote -v

origin https://github.com/example/example-repo.git (fetch)

origin https://github.com/example/example-repo.git (push)

以上是创建了 "origin" 别名并将其 URL 设置为 "https://github.com/example/example-repo.git" 后的示例输出。

问题三十一

解释以下git命令。git push origin feature/东方不败改进

答: 用于将本地名为 "feature/东方不败改进"的分支，提交推送到名为 "origin" 的远程库中。

 git push  表示将本地分支的提交推送到远程库。

 origin  是远程库的别名，通常用作默认远程库的别名。

 feature/东方不败改进  是本地分支的名称，您希望将此分支的提交推送到远程库中的相应分支。

问题三十二

如果远程库没有feature/东方不败改进的分支。git push origin feature/东方不败改进，执行后会怎么样

如果远程库没有名为feature/东方不败改进的分支，那么执行git push origin feature/东方不败改进命令后，会出现错误提示，提示没有找到该分支。Git会尝试将本地分支推送到远程仓库相同名称的分支。如果远程仓库中没有该分支，则无法推送成功。

问题三十三

怎么避免该问题。

答:要避免这个问题，您可以在执行git push命令之前，先使用git branch命令确认本地是否存在名为feature/东方不败改进的分支，并且该分支已经与远程仓库进行了关联。

如果本地分支不存在，可以使用git checkout -b feature/东方不败改进命令创建并切换到该分支。然后使用git push -u origin feature/东方不败改进命令将该分支推送到远程仓库，并与远程仓库的同名分支进行关联。

如果本地分支已存在但与远程仓库未关联，可以使用git branch --set-upstream-to=origin/feature/东方不败改进命令将本地分支与远程仓库的同名分支进行关联，然后再执行git push命令进行推送。

在确认本地分支存在且与远程仓库关联后，再执行git push origin feature/东方不败改进命令，就能成功地将本地分支推送到远程仓库。

问题三十四

git push origin master，会出现因为远程库没有分支，报错的情况吗？

答:在执行 git push origin master 命令时，如果远程仓库并没有名为 master 的分支，那么会出现报错的情况。

常见的报错信息可能是：

error: src refspec master does not match any

error: failed to push some refs to 'origin'

这通常表示本地的 master 分支找不到对应的远程分支，可能是因为远程仓库中并没有创建名为 master 的分支。

解决该问题的方法是，先创建远程仓库中的 master 分支，可以使用以下命令：

git checkout -b master

git push origin master

如果远程仓库中已经存在 master 分支，那么可能是某些原因导致本地和远程仓库中的分支不同步，可以使用以下命令来更新本地分支：

git fetch origin

git merge origin/master

以上命令会从远程仓库中拉取最新的 master 分支并合并到本地的 master 分支。然后再执行 git push origin master 命令即可推送本地分支到远程仓库。

6.5 1000大卡

问题三十五

github创建完远程库，它不会自动有一个master分支吗

答:是的，当使用Github等远程仓库平台创建一个新的仓库时，通常会自动为其创建一个默认分支，且该默认分支一般命名为 master 。在执行 git push origin master 命令时，会将本地的 master 分支推送到远程仓库的 master 分支。

但是，也有一些远程仓库平台，例如GitLab，在创建仓库时不会自动创建 master 分支，而是使用其他名称作为默认分支。

所以，在执行 git push origin master 之前，确保远程仓库已经创建了 master 分支或者使用了其他名称的默认分支。否则会出现报错信息，表示找不到对应的远程分支。如果你在使用Github创建仓库时遇到了类似的问题，可以检查远程仓库设置中是否有 master 分支。

6.5小时 一千大卡

问题三十六

git将远程库更新的代码和本地做同步

答:要将远程库更新的代码与本地库进行同步，可以按照以下步骤进行操作：

首先，确认你当前在本地库的工作目录中。可以使用 git status 命令来查看当前的工作状态。

使用 git fetch 命令从远程库中获取最新的代码更新。这个命令会将远程库中的更新代码下载到本地的一个临时分支中。

使用 git merge 命令将刚刚下载的代码合并到本地分支中。运行 git merge origin/branch ，其中 branch 是远程分支的名称。

另一种方法是使用 git pull 命令，它其实是 git fetch 和 git merge 两个步骤的简化版本。运行 git pull origin branch ，其中 branch 是远程分支的名称。

请注意，如果在合并过程中存在冲突，需要手动解决冲突。冲突解决后，使用 git add 命令将文件标记为已解决，并运行 git commit 命令来完成合并。

问题三十七 6.5小时，一千大卡

github，如何邀请团队伙伴成为我创建远程库的成员

答:要邀请团队伙伴成为你在GitHub上创建的远程库的成员，你需要遵循以下步骤：

在GitHub上登录你的账号并进入你创建的远程库页面。

点击页面上方的 "Settings"（设置）选项卡。

在左侧导航栏中选择 "Manage access"（管理访问权限）选项。

在页面的右上方，点击 "Invite a collaborator"（邀请协作者）按钮。

在弹出的窗口中，输入你想邀请的团队伙伴的GitHub用户名、电子邮件地址或姓名。

GitHub会为你输入的信息提供匹配结果，选择你要邀请的团队伙伴。

可以选择添加一个团队或组织来定义邀请的权限范围，或者直接选择 "Write"（写入）或 "Read"（只读）权限。

点击 "Add [username] to [repository name]"（将 [用户名] 添加到 [远程库名称]）按钮，邀请将被发送给团队伙伴。

GitHub会发送邀请邮件给被邀请的团队伙伴，他们需要在邮件中点击相应的链接来接受邀请。

请注意，被邀请的团队伙伴需要有一个GitHub账号才能接受你的邀请。另外，你可能需要相应的权限才能邀请他人成为远程库的成员。

问题三十八

邀请成功后，他可以做什么

答:一旦你成功邀请团队伙伴成为你在GitHub上创建的远程库的成员，他们将获得相应的权限，可以进行以下操作：

读取代码：被邀请成员可以查看远程库中的所有代码文件和文件夹，包括历史提交记录。

编辑和推送代码：具有写入权限的被邀请成员可以修改代码、创建新的分支、提交和推送更改到远程库。

管理问题和合并请求：被邀请成员可以查看和管理与远程库相关的问题、提出合并请求、参与讨论和审核代码。

发布版本：具有相应权限的被邀请成员可以在远程库中创建和发布新的版本，以及管理版本的发布。

总之，被邀请的团队伙伴将可以与你协作开发远程库中的代码并参与项目的管理。他们的权限取决于你在邀请时为他们分配的权限级别。

问题三十九

邀请成功后，他可以做什么

答:一旦你成功邀请团队伙伴成为你在GitHub上创建的远程库的成员，他们将获得相应的权限，可以进行以下操作：

读取代码：被邀请成员可以查看远程库中的所有代码文件和文件夹，包括历史提交记录。

编辑和推送代码：具有写入权限的被邀请成员可以修改代码、创建新的分支、提交和推送更改到远程库。

管理问题和合并请求：被邀请成员可以查看和管理与远程库相关的问题、提出合并请求、参与讨论和审核代码。

发布版本：具有相应权限的被邀请成员可以在远程库中创建和发布新的版本，以及管理版本的发布。

总之，被邀请的团队伙伴将可以与你协作开发远程库中的代码并参与项目的管理。他们的权限取决于你在邀请时为他们分配的权限级别。

问题四十

github，有很多同名远程库，如何快速找到某个人的远程库

答:要快速找到某个人的远程库，你可以按照以下步骤进行：

在GitHub的首页上方搜索栏中，输入该人的用户名或组织名，并按下回车键。

在搜索结果页面的左侧选择“Repositories”选项卡，以过滤出只显示该用户或组织下的远程库。

浏览搜索结果，查找目标远程库。你可以通过查看库的名称、描述以及库所属者的用户名来确定目标库。

如果你已经知道目标库的名称，可以直接在搜索栏中输入目标库的名称，然后在搜索结果中找到该库。

问题四十一

github，找到别人的远程库，点击fork，作用是什么

答:在GitHub上，点击"fork"按钮可以将别人的远程库复制到自己的GitHub账号下的仓库中，创建一个与原仓库完全独立但内容相同的新仓库。这个操作的作用主要有以下几点：

获取原仓库的拷贝：通过fork，你可以创建一个包含原仓库所有文件和历史记录的仓库的拷贝。这样你就可以在自己的账号下对该仓库进行修改和管理。

进行个人修改：fork后，你可以自由地在自己的拷贝仓库中进行修改、添加和删除文件，而不会影响到原仓库。这为你提供了一个独立的开发空间。

提交贡献：当你在fork后的仓库中进行了一些修改或者改进之后，你可以将这些修改提交回原仓库。通过提出一个pull request来通知原始项目维护者，请求他们审查你的修改并将其合并到原仓库中。这样你就可以为原始项目做出贡献。

总之，点击"fork"按钮可以创建一个与原仓库相独立的个人拷贝，让你自由地修改和管理代码，并且可以向原仓库提交贡献。

问题四十二

github，fork并修改代码后，如何使用pull request

答:使用pull request的基本流程如下：

Fork仓库：在GitHub上找到你想贡献的项目仓库，点击右上角的"fork"按钮，将其复制到自己的GitHub账号下的仓库。

Clone仓库：在你自己的GitHub账号下找到这个fork的仓库，点击绿色的"Code"按钮，复制仓库的URL。然后在本地的终端中使用git命令克隆该仓库，例如：

git clone <仓库URL>

创建分支：切换到克隆下来的仓库目录，使用git命令创建一个新分支来添加你的修改，

git checkout -b <分支名>

进行修改：在新分支上进行你的修改和编辑，可以新增、删除、修改文件等。

提交修改：在本地完成你的修改后，使用git命令来提交你的修改，例如：

git add .

git commit -m "描述你的修改"

推送分支：将你的修改推送到你的GitHub账号下的仓库，例如：

git push origin <分支名>

提交pull request：在你自己GitHub账号下的仓库中，进入你刚刚推送的分支页面，点击"New pull request"按钮，填写相关信息，描述你的修改并提交pull request。

等待审批：等待原始项目维护者审查你的修改，提出可能的改进或建议，并决定是否接受你的pull request。

通过以上步骤，你就可以使用pull request向原始项目的维护者提交你的修改，希望将其合并到原仓库中。

问题四十三

git命令，如何获得ssh免密登录的公钥

答:使用以下步骤可以获取你的SSH公钥：

打开终端或命令行窗口。

输入以下命令以生成SSH密钥对：

ssh-keygen -t rsa -C "your\_email@example.com"

将 your\_email@example.com 替换为你的邮箱地址。

系统会提示你选择密钥文件的保存路径和设置密码。可以按回车键接受默认路径和留空密码来生成密钥。

生成成功后，可以在默认路径（ ~/.ssh/ ）下找到两个文件： id\_rsa 和 id\_rsa.pub ，其中 id\_rsa 是私钥， id\_rsa.pub 是公钥。

使用以下命令来显示和复制SSH公钥：

cat ~/.ssh/id\_rsa.pub

 终端中会显示SSH公钥的内容，类似于以下示例：

ssh-rsa AAAAB3NzaC1y... your\_email@example.com

问题四十四

解释以下命令。ssh-keygen -t rsa -C "your\_email@example.com"

答: ssh-keygen ：这是一个用于生成SSH密钥对的命令。它是OpenSSH提供的一个工具，用于生成和管理SSH密钥。

 -t rsa ：这是 ssh-keygen 命令的一个选项，用于指定生成的密钥类型。在这个命令中，我们使用RSA算法生成密钥。

 -C "your\_email@example.com" ：这是 ssh-keygen 命令的另一个选项，用于指定密钥的注释信息。在这个命令中，我们使用 your\_email@example.com 作为注释信息。

综合起来， ssh-keygen -t rsa -C "your\_email@example.com" 命令的作用是生成一个RSA算法的SSH密钥对，并将指定的电子邮件地址作为注释信息添加到密钥中。这个密钥对包括一个私钥（ id\_rsa ）和一个公钥（ id\_rsa.pub ），它们分别在本地机器上用于身份验证和远程服务器上用于验证你的身份。

问题四十五

github，把生成好的公钥放在哪里

答:当你使用GitHub时，你可以将生成的公钥放在你的GitHub帐户中的SSH设置中。这样，你就可以通过SSH密钥进行安全的代码推送和拉取操作。

下面是一些在GitHub上添加公钥的步骤：

在你的计算机上生成SSH密钥对。你可以使用命令行工具（如OpenSSL）或SSH客户端（如PuTTY）来生成密钥。确保生成的密钥对包括公钥和私钥。

登录到你的GitHub帐户。点击右上角的个人头像，选择"Settings"（设置）。

在设置页面中，选择"SSH and GPG keys"（SSH和GPG密钥）选项卡。

点击"New SSH key"（新的SSH密钥）。

在"Title"（标题）字段中，可以为你的密钥取一个描述性的标题（如"Work Laptop"或"Personal Desktop"）。

在"Key"（密钥）字段中，将你的公钥内容粘贴进去。你可以在生成的公钥文件中找到这个内容。注意要将整个公钥粘贴到"Key"字段中。

点击"Add SSH key"（添加SSH密钥）。

你现在已经成功将公钥添加到你的GitHub帐户中。

当你使用SSH进行与GitHub的交互时，GitHub将使用你的公钥进行身份验证。这将使你能够在不输入用户名和密码的情况下进行代码操作

问题四十六

github，有了公钥后，有哪些命令可以免登

答:在GitHub上使用SSH密钥后，你可以使用以下命令来免登录操作：

克隆存储库：使用SSH URL克隆一个GitHub存储库时，将使用你的SSH密钥进行身份验证，而不是要求你输入用户名和密码。使用以下命令：

git clone git@github.com:username/repository.git

 其中， username 是你的GitHub用户名， repository 是你想要克隆的存储库的名称。

推送更改：当你使用SSH密钥进行推送操作时，不需要输入用户名和密码。使用以下命令将代码推送到你的远程存储库：

git push git@github.com:username/repository.git

 注意：确保你正在使用SSH URL进行推送，而不是HTTP URL。

拉取更改：使用SSH密钥进行代码拉取操作时，不需要输入用户名和密码。使用以下命令从远程存储库拉取代码更改：

git pull git@github.com:username/repository.git

 与推送操作类似，确保你正在使用SSH URL进行拉取操作。

这些命令只是在你已经成功将SSH公钥添加到你的GitHub帐户后才能免登录。请确保你已按照先前提到的步骤来添加公钥并验证它是否正常工作。

问题四十六

确保你正在使用SSH URL进行拉取操作，怎么确保

答:检查你已经成功设置SSH密钥并将其添加到你的GitHub帐户中。你可以按照之前提到的步骤来完成这个过程。

打开你要拉取代码的GitHub存储库页面。

点击绿色的"Code"按钮，点击剪贴板图标按钮来复制SSH URL。

在命令行中使用 git 命令进行拉取操作，将复制的SSH URL粘贴到命令后面。

git pull git@github.com:username/repository.git

 将 username 替换为你的GitHub用户名， repository 替换为你要拉取的存储库名称。

确保你执行上述步骤，并在使用 git pull 命令时使用正确的SSH URL，这将确保你使用SSH进行拉取操作，并实现免登录。

问题四十七

git，如何创建ignore忽略文件，同时把路径放到config文件中，使其生效

答:要在Git中创建忽略文件（.gitignore），只需在项目根目录下创建一个名为“.gitignore”的文本文件，并在文件中指定要忽略的文件或文件夹的规则。

例如，如果你想忽略所有“.txt”文件，你可以在“.gitignore”文件中添加以下内容：

\*.txt

 如果你想忽略一个文件夹（例如“logs”文件夹）

logs/

 如果你想忽略特定的文件（例如“config.txt”文件）

config.txt

你可以使用以下命令来指定一个全局的Git忽略文件路径：

git config --global core.excludesfile /path/to/your/.gitignore

 将上述命令中的“/path/to/your/.gitignore”替换为你自己的实际路径。

问题四十八 5.5小时 一千大卡

idea如何配置git的exe文件路径

答:打开 IntelliJ IDEA，点击菜单栏的 "File"（文件），然后选择 "Settings"（设置）。

在弹出的设置窗口中，找到左侧导航栏中的 "Version Control"（版本控制）选项，在下方展开的列表中选择 "Git"。

在右侧的界面中，可以看到 "Path to Git executable"（Git 可执行文件路径）的输入框。

如果你已经在电脑中安装了 Git，并且 Git 的可执行文件已经被添加到系统的环境变量中，那么 IntelliJ IDEA 会自动检测到 Git 的路径。你可以在此处看到自动检测到的路径，并且无法对其进行编辑。

问题四十九 5.5小时 一千大卡

idea创建一个项目后，如何初始化git，进行add和commit操作

答:在 IntelliJ IDEA 中创建一个新项目或者打开一个已存在的项目。

打开菜单栏的 "VCS"（版本控制），然后选择 "Enable Version Control Integration"（启用版本控制集成）。

在弹出的对话框中，选择 "Git" 作为版本控制系统，并点击 "OK"。

在底部的 Version Control 工具窗口中，可以看到项目文件的变化。右键单击项目文件或文件夹，选择 "Git"，然后选择 "Add"（添加文件到暂存区）。

选择要添加到暂存区的文件，可以选择单个文件或者整个文件夹。

在底部左下角的 Version Control 工具窗口中，你将看到被添加到暂存区的文件出现在 "Default"（默认）选项卡下。

右键点击被添加到暂存区的文件，选择 "Commit"（提交）。

在弹出的提交窗口中，输入提交的消息，描述此次提交的内容。

点击 "Commit" 按钮，完成提交操作。

问题五十 5.5小时 一千大卡

idea，如何使用version control，切换到最新一次提交之前的提交

答:确保你已经启用了版本控制集成并成功进行了一次提交。

在底部的 Version Control 工具窗口中，你将看到提交历史记录。

找到最新的提交的快照，并右键单击它。

在弹出的上下文菜单中，选择 "Reset | Reset Current Branch to Here"（重置当前分支到这里）。

在弹出的对话框中，选择 "Soft"（保留更改）或 "Mixed"（保留更改并将更改添加到暂存区）选项，然后点击 "Reset"（重置）按钮。

此时，你的代码将会回滚到最新一次提交之前的状态。

问题五十一

idea，如何使用version control的log功能，切换到最新一次提交之前的提交

答:确保你已经启用了版本控制集成，并成功进行了一次提交。

在底部的 "Version Control" 工具窗口中，点击 "Log"（日志）选项卡，你将看到所有的提交记录。

找到你想要切换到的特定提交，并右键单击它。选择 "Checkout Revision"（切换到此版本），你的代码将会回滚到所选的提交版本。

问题五十二

idea，如何给项目创建分支

答:点击顶部菜单中的“VCS”。

选择“Git”下的“Branches”选项。

在弹出的窗口中，点击右上角的“+ New Branch”按钮。

在弹出的对话框中，输入分支的名称。

如果您想基于现有分支创建出的分支，可以选择在“Base branch”下拉菜单中选择相应的分支。

点击“Create”按钮来创建新的分支。

这样，IntelliJ IDEA就会为您的项目创建一个新的分支，而不需要手动输入git命令。

问题五十三

不用手动输入git命令，在idea，选择项目名右键选择git，选给项目创建分支

答:在IDEA的项目导航窗口中选择您的项目名称。

右键点击项目名称，然后选择"Git"。

在弹出的上下文菜单中，选择repository 和"Branches"。

在弹出的分支窗口中，点击右上角的"+"按钮。

在弹出的对话框中，输入分支的名称，然后点击"Create"按钮。

问题五十四

idea还可以点击右下角的git，选择new branches，创建分支

答:打开您的项目。

点击右下角的Git图标，以打开Git工具窗口。

在Git工具窗口中，切换到"Branches"选项卡。

在分支列表上方的工具栏中，点击"New Branch"按钮（一个图标为"+"的按钮）。

在弹出的对话框中，输入分支的名称，然后点击"OK"按钮。

问题五十五

在idea，选择右下角git，合并分支hotfix到master

答:先进入 "master" 分支，点击底部工具栏的 "Git" 图标，选择要合并的hotfix分支），右键单击它并选择 "Merge Into Current"（合并到当前分支）。

完成上述步骤后，分支 "hotfix" 将被合并到 "master" 分支中。您可以在Git窗口的 "Log"（日志）选项卡中查看合并结果，并将更改推送到远程Git仓库。请注意，在合并期间可能会出现冲突，您可能需要解决这些冲突后才能成功合并分支。

问题五十六

idea，合并分支hotfix到master如何解决冲突，

答:如果存在冲突，IDEA会弹出冲突解决工具窗口

在冲突解决器中，您可以看到冲突的两个版本以及合并结果。您可以手动编辑代码，选择保留哪个版本或者自己修改代码。

在冲突解决器中，您可以点击 "apply"（解决）按钮，标记冲突已解决。

逐个解决所有冲突后，再次打开右下角的Git工具窗口，确保所有冲突已解决。

问题五十七

通过IDEA设置使用GitHub账号并使用token登录

答：1. 选择 "Settings"（设置），version control 选择github

6. 点击 ➕（新建）按钮。

7. 在弹出的窗口中，输入您的GitHub用户名和token信息。确保token的安全性，不要将其暴露给不信任的人。

8. 输入完成后，点击 "OK" 保存设置。

9. 完成上述步骤后，您应该能够在IDEA中通过GitHub身份验证并访问GitHub上的项目。

请注意，使用token登录可以简化登录过程，但请确保妥善保管您的token信息，以防止安全风险。

问题五十八

token如何生成

答：打开GitHub的网站并登录到您的账号。

转到用户设置页面，点击页面右上角的头像图标，然后选择“Settings”（设置）选项。

在设置页面的左侧菜单中，选择“Developer settings”（开发者设置），选择“Personal access tokens”（个人身份访问令牌）。点击“Generate new token”（生成新令牌）按钮。

输入一个描述性的令牌名称，并选择要授予该令牌的权限。

在选择完权限后，点击页面底部的“Generate token”（生成令牌）按钮。

生成的令牌将会显示在页面上。请注意，在离开该页面后您将无法再次查看这个令牌。

问题五十九

不使用git命令，idea如何把项目传到github中

答：打开IntelliJ IDEA，选择要上传的项目。

在IntelliJ IDEA的顶部菜单栏中，选择"VCS"（版本控制系统） -> "Import into Version Control"（导入到版本控制）-> "Share Project on GitHub"（分享项目到GitHub）。

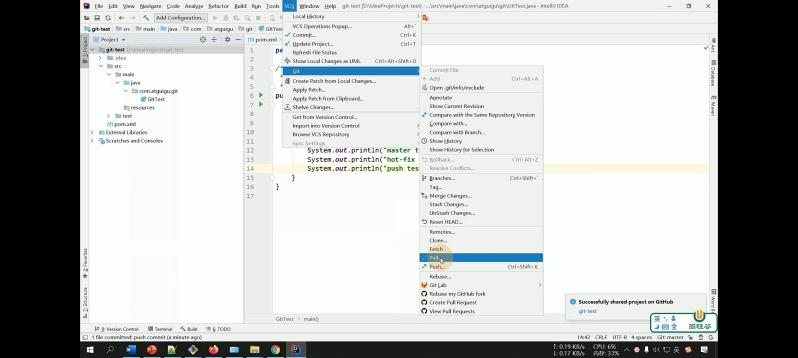
输入GitHub仓库的名称，给远程库链接起个别名。

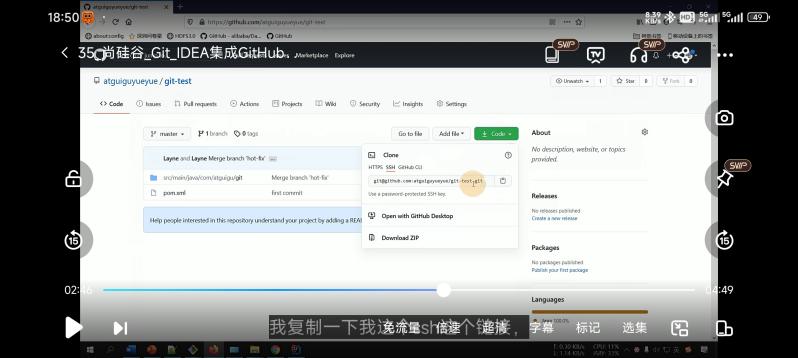
IDEA会自动创建一个新的GitHub仓库，并将项目上传到该仓库中。

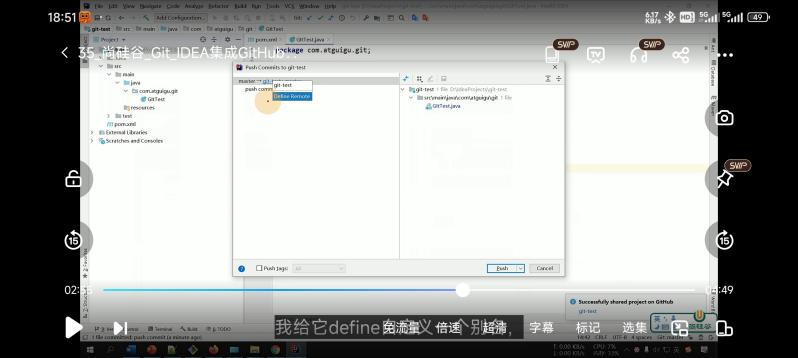
问题六十

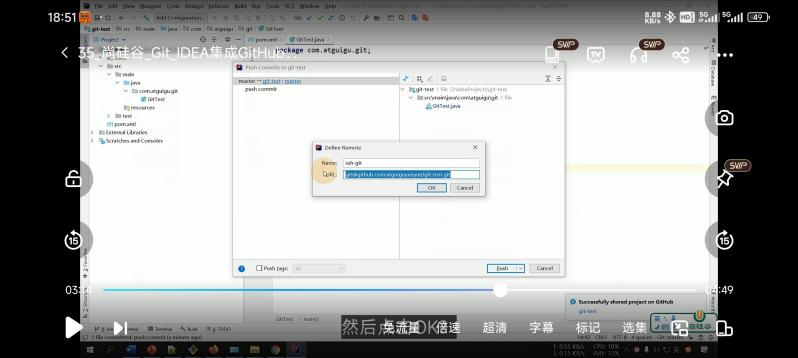
不使用git命令，idea如何push已经commit的代码到github中

答：





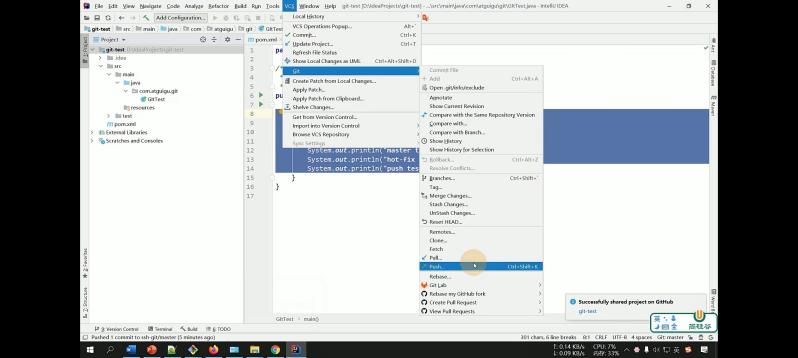


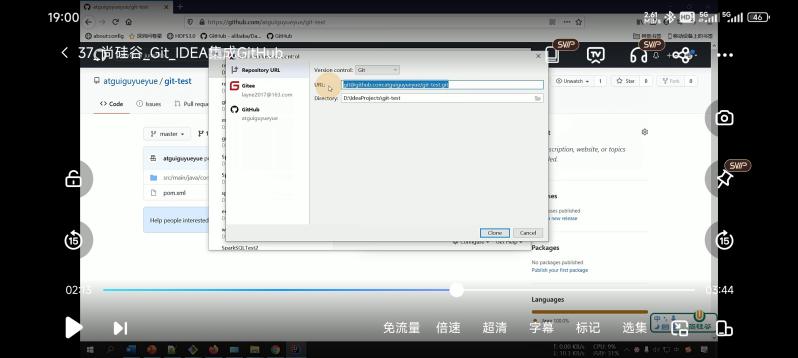
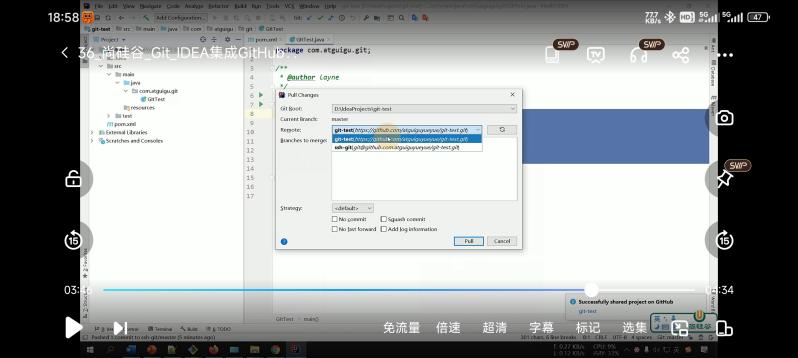


问题六十

idea如何pull别人更新到github远程库的代码

答：

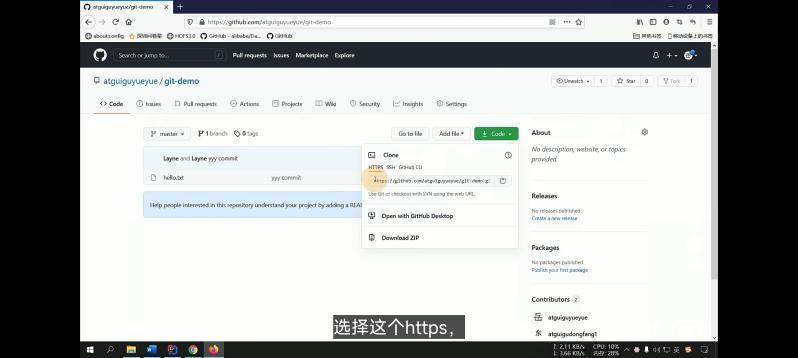




问题六十一

新同事如何通过idea，clone公司远程库

答：



问题六十二

gitlab如何在cent os7.0安装

答：在CentOS 7.0上安装GitLab，

更新系统并安装依赖项：

sudo yum update

sudo yum install curl openssh-server postfix

启动并设置Postfix服务：

sudo systemctl Enable postfix

sudo systemctl start postfix

下载并安装GitLab Repository：

curl -LO https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh

sudo bash script.rpm.sh

安装GitLab CE（Community Edition）：

sudo yum install gitlab-ce

配置并启动GitLab：

sudo gitlab-ctl reconfigure

 完成上述步骤后，你可以在浏览器中访问 http://<服务器IP> 或者 http://<服务器域名> 来访问GitLab的安装界面。