

# C++基础课程

## 第零章——工欲善其事必先利其器

### 0. 提前说明

- 本课程均以 windows 11 系统作为示例，若为 Linux 或者 Macos 可另外课外提问或自己查阅资料。
- 个人建议，少用百度，能则 Google，不行的话 Bing 体验也不错。

## 第一节——Git及Github/Gitee的使用

### 0.0 自主学习

你也可以通过GitHub官方文档[快速入门](#)或者其他有关网站比如<https://www.atlassian.com/zh/git/tutorials>来自主学习。

如果不想在本地使用Git，你也可以下载并安装[GitHub Desktop](#)客户端，请参照GitHub官方文档[GitHub Desktop 使用入门](#)自主学习，本课程对此不作教学。

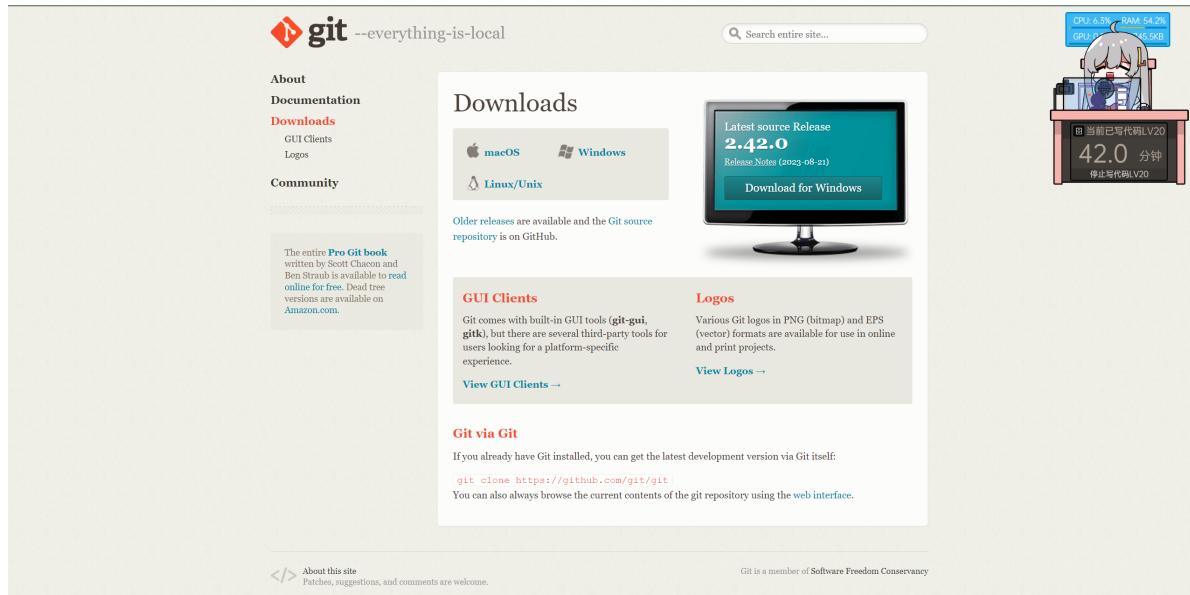
### 0.1 什么是Git？

1. **版本管理：** Git允许开发人员追踪项目的不同版本和修改，以便可以随时回退到之前的版本，查看修改历史，比较不同版本之间的差异，以及解决冲突。这对于协作开发和维护软件项目非常重要的。
2. **协作：** Git是分布式版本控制系统，允许多个开发人员同时在同一项目上工作。开发人员可以在各自的本地仓库上工作，然后将其更改推送到共享的远程仓库。这使得团队协作更加高效，每个人都可以独立工作，而不会干扰其他人的工作。
3. **分支管理：** Git使得创建、合并和管理分支变得非常容易。开发人员可以创建新分支来开展新的功能开发或修复问题，而不会影响主分支。一旦工作完成，可以将分支合并回主分支。
4. **备份和恢复：** Git提供了对项目的完整历史记录的备份。即使在本地计算机出现故障或远程服务器遇到问题时，您仍可以从备份中恢复项目状态。
5. **分布式架构：** Git的分布式性质使得每个开发人员都有整个项目的拷贝，这降低了对中央服务器的依赖。即使中央服务器出现故障，开发人员仍然可以继续工作。
6. **开源社区：** Git是许多开源项目的首选版本控制系统，因此学习Git可以更容易地参与和贡献开源项目。
7. **跨平台支持：** Git可在多种操作系统上运行，包括Windows、Linux和macOS，因此可以满足不同开发环境的需求。
8. **顺应潮流：** 聪明的开发人员应该顺应潮流。Git 正在被越来越多的知名公司和开源项目所使用，如 Ruby On Rails, jQuery, Perl, Debian, Linux 内核等等。拥有一个大型的用户群体是一个很大优势，因为往往会有许多系统去推动他的发展。大量的教程，工具和服务，这让 Git更加具有吸引力。

### 1. 安装Git

进入官网，根据自己的系统选择下载即可，如果下载太慢可以下载群文件里的。

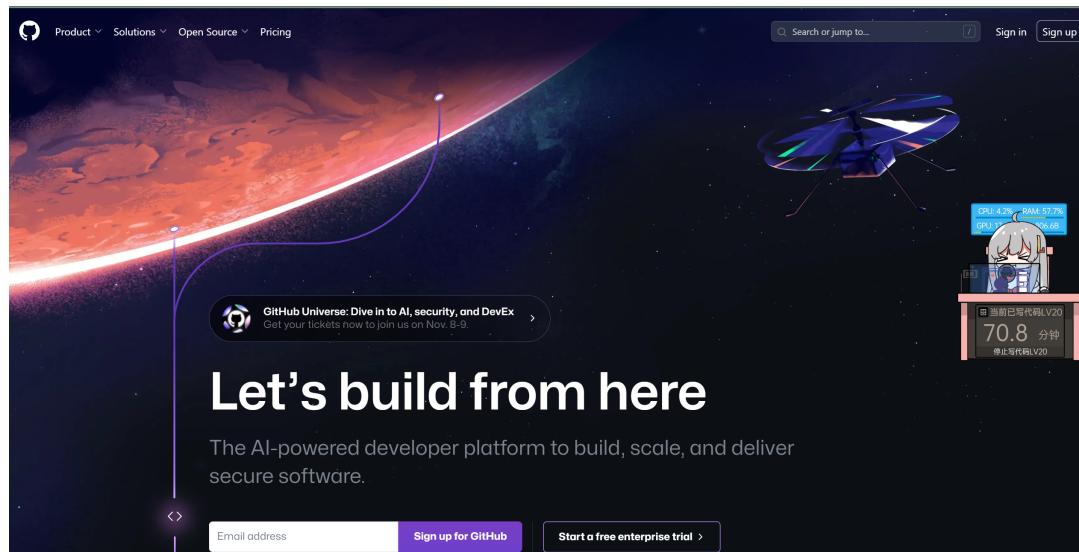
<https://git-scm.com/downloads>



注意在 `Select Components` 时一定要选上 `git Bash Here` (其实会自动勾选)。

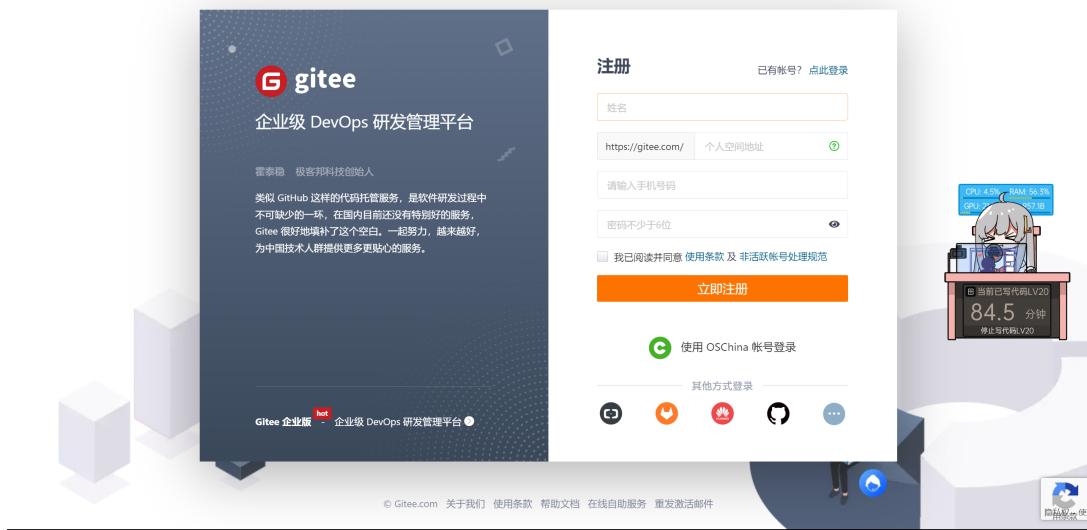
## 2. 创建GitHub/Gitee账户

- Github账户的创建
  - 如果你能创建GitHub账户，那么大概率你能翻墙了，想必也不用多教了，如果你不能翻墙，那你创建GitHub账户也没什么用。



- Gitee账户的创建

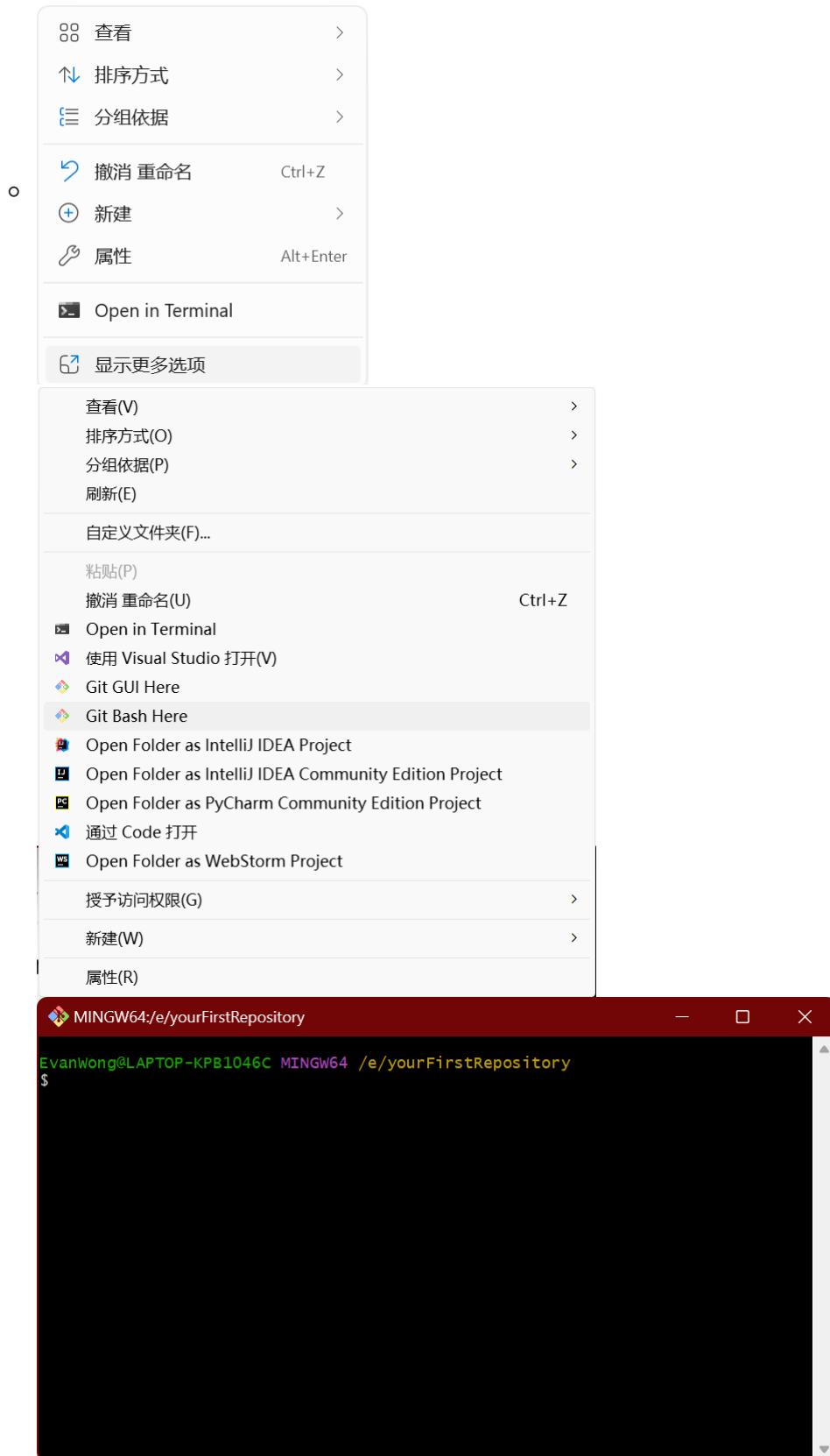
- 官网自行注册即可



### 3. Git基础命令的使用

接下来请紧跟步骤尝试自己创建并上传一个仓库，在此过程中熟悉Git的基础使用。

- 配置信息，这些信息将出现在你的 Git 提交中
  - git config --global user.name "Your Name"
  - git config --global user.email "[youremail@example.com](mailto:youremail@example.com)"
- 创建仓库
  - 创建一个新的文件夹，右键选择 显示更多选项 - **Git Bash Here**



- 直接输入 `git`，你可以看到能使用的全部指令及用处

```

○ EvanWong@LAPTOP-KPB1046C MINGW64 /e/yourFirstRepository
$ git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  diff      Show changes between commits, commit and working tree, etc
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  commit   Record changes to the repository
  merge    Join two or more development histories together
  rebase   Reapply commits on top of another base tip
  reset   Reset current HEAD to the specified state
  switch   Switch branches
  tag      Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch   Download objects and refs from another repository
  pull    Fetch from and integrate with another repository or a local branch
  push    Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

```

- 输入 `git init` 以创建一个空的git repo。
- 创建一个 `README.md`，随便向其写入内容，使用 `git add` 指令将其添加到暂存区 (Staging Area) 中。你可以使用 `git add filename` 将单个指定文件添加到git repo中，或者使用 `git add .` 将所有新修改文件添加到暂存区 (Staging Area) 中，或者使用 `git add directory/` 添加指定目录下的所有修改文件到暂存区 (Staging Area) 。
- 使用 `git commit` 命令将在暂存区 (Staging Area) 中的文件快照保存到本地仓库中，每个 `git commit` 创建的提交对象都有一个唯一的哈希值，用于标识版本。这使得你可以随时回滚到先前的版本，比较不同版本之间的差异，以及追踪代码的演变。具体用法为 `git commit -m "commit message"`，提交信息用于描述本次提交的目的，这十分有用。
  - 你也可以使用 `-a` 选项来跳过 `git add` 步骤，将工作目录中的所有更改（包括已被跟踪的文件）添加到暂存区并提交，但要小心使用它，因为它不会包含新添加的未跟踪文件。具体用法为 `git commit -a -m "Commit message"`
- 你可以使用 `git status` 显示当前工作目录和暂存区的状态，以便你了解哪些文件已被修改、添加到暂存区或尚未被跟踪。接下来修改 `README.md` 的内容，创建 `casually1.txt`, `casully2.txt` 两个文件，将 `casually1.txt` 添加到暂存区。此时使用 `git status`，你就可以查看到三种状态 `Changes to be committed`, `Changes not staged for commit`, `Untracked files`。
  - **未跟踪的文件(Untracked files)**: 这是工作目录中未被 Git 跟踪的文件。这些文件不在 Git 的版本控制之下。
  - **已修改的文件(Changes not staged for commit)**: 这是工作目录中已被修改但尚未添加到暂存区的文件。

- **已暂存的文件(Changes to be committed)**: 这是已经添加到暂存区，等待被提交的文件。

```
EvanWong@LAPTOP-KPB1046C MINGW64 /e/yourFirstRepository (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:  casually.1

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

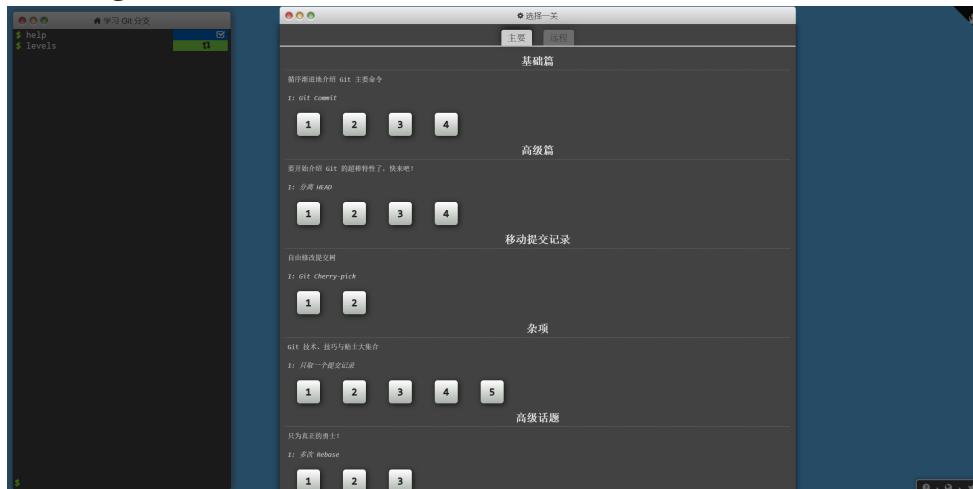
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    casually.2
```

- 使用 `git log` 指令打印仓库提交日志。

```
EvanWong@LAPTOP-KPB1046C MINGW64 /e/yourFirstRepository (master)
$ git log
commit 6aa376ba1338562aa6e5c1e0e8362556bd4c4371 (HEAD -> master)
Author: EvanWong <yufenghuang09@gmail.com>
Date:   Sun Oct 15 11:23:45 2023 +0800

  first commit
```

- `git` 分支指令可能暂时使用较少，本课程暂时不涉及。
- 你可以在网站[https://learngitbranching.js.org/?locale=zh\\_CN](https://learngitbranching.js.org/?locale=zh_CN)以游戏的方式学习并巩固你的`git`指令。



## • 使用SSH完成 Git 与 GitHub/Gitee 的绑定

- 在 `Git Bash` 中输入 `ssh`，可以查看指令及参数。

```
EvanWong@LAPTOP-KPB1046C MINGW64 /e/yourFirstRepository (master)
$ ssh
usage: ssh [-46AaCfGgKkMNsTtVvXxYy] [-B bind_interface]
           [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
           [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
           [-i identity_file] [-J [user@]host[:port]] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] destination [command [argument ...]]
```

- 输入 `ssh-keygen -t rsa` 指定 RSA 算法生成密钥，然后敲三次回车键。
- 使用 `cat ~/.ssh/id_rsa.pub` 获取公钥。
- 如果你是用的是 GitHub，在 `Settings -- SSH and GPG keys` 处选择 `New SSH key`，输入 `title` 和 `key` 即可，`title` 可随意。

The screenshot shows the GitHub Settings interface under the 'SSH and GPG keys' section. On the left, there's a sidebar with various account settings like Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, Sessions, and SSH and GPG keys (which is selected). The main area is titled 'SSH keys' and contains a table of keys. Each key entry includes a key icon, the email address, the type (SSH), the date it was added, the last used date, and a 'Delete' button.

This screenshot shows the 'Add new SSH Key' form. The sidebar on the left is identical to the previous one. The main form has fields for 'Title' (a text input), 'Key type' (a dropdown menu set to 'Authentication Key'), and a large text area for 'Key' containing placeholder text about supported formats. A green 'Add SSH key' button is at the bottom right.

- 如果你使用的是Gitee，在设置-安全设置-SSH公钥处添加。

The screenshot shows the Gitee Settings interface under the 'SSH公钥' (SSH Keys) section. The sidebar on the left includes options like 消息中心, 基本设置, 安全设置, and SSH公钥 (which is selected). The main area shows a list of existing SSH keys with columns for title, key, and expiration date. Below this is a '添加公钥' (Add Key) form with fields for '标题' (Title) and '公钥' (Key), and a note explaining the key format. A large text area for pasting the key is provided, along with a '保存' (Save) button at the bottom.

- 将本地仓库推送到 Github/Gitee

- 在 GitHub/Gitee 创建新仓库

- GitHub可在主页或者Repositories处找到 new 按钮

The screenshot shows the Gitee Dashboard. At the top, there's a navigation bar with a menu icon, a user icon, and the word "Dashboard". Below this is a section titled "Top Repositories" with a "New" button. A search bar contains the placeholder "Find a repository...". The main area displays a list of repositories, with one repository named "211LabLectures" shown in detail. This repository has 16 stars and was updated yesterday. Below the repository list, there's a "gitee" logo with the text "开源软件 企业版". On the right, there's a user profile for "HUANG Yuf...", showing activity statistics: 仓库 (4), Pull Requests (0), Issues (0), 代码片段 (0), and 我的星选集 (2). A red arrow points from the "New" button to the "+ 新建" button in the "我活跃的仓库" section.

Top Repositories

New

Find a repository...

EvanWonghere

Overview Repositories 16 Projects Packages Stars 37

211LabLectures Public  
The lectures of 211 robotic lab  
Updated yesterday

Star

gitee 开源软件 企业版

HUANG Yuf... 个人主页

仓库 4  
Pull Requests 0  
Issues 0  
代码片段 0  
我的星选集 2

我的企业/高校/组织

■ Gitee可在工作台（主页）处找到新建按钮

你还没有加入组织/企业  
了解 Gitee 企业版 或 创建免费企业版

我活跃的仓库 + 新建

搜索与我相关的仓库

HUANG Yufeng/MathematicsMode...  
HUANG Yufeng/Research-of-Bhua...  
HUANG Yufeng/hahahah  
HUANG Yufeng/Wordle

最近参与

全部 Issues Pull Request

你最近没有活跃的Issue/Pull request

- 推送本地仓库，执行 GitHub/Gitee 给出的官方指令即可

The screenshot shows the GitHub/Gitee quick setup interface. It includes sections for setting up GitHub Copilot, adding collaborators, and a 'Quick setup' section. The 'Quick setup' section contains three command-line snippets:

- ...or create a new repository on the command line**

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:EvanWonghere/test.git
git push -u origin main
```

- ...or push an existing repository from the command line** (highlighted with a red arrow)

```
git remote add origin git@github.com:EvanWonghere/test.git
git branch -M main
git push -u origin main
```

- ...or import code from another repository**

Below these snippets is a Gitee landing page screenshot. It shows the Gitee logo, navigation links (开源软件, 企业版, 高校版, 私有云, 博客, 我的), and a search bar. The main content area displays the 'Quick setup' guide again, with the '...or push an existing repository from the command line' section highlighted.

## ● 推送本地更新

- 当你更改了本地仓库之后，需要将本地仓库同步到 GitHub/Gitee 上。具体步骤为

```
1 | git add filename / git add .
2 | git commit -m "Updated nothing"
3 | git pull
4 | git push
```

### ○ git pull:

`git pull` 用于从远程仓库拉取（获取）更新并将其合并到你的当前本地分支。它执行以下操作：

- 检查当前所在分支。
- 从远程仓库获取最新的提交。
- 将这些提交合并到你的当前分支。

基本语法如下：`git pull remote_repo "branch"`。如果不指定远程仓库和分支，`git pull` 将默认拉取与当前本地分支关联的远程分支的更新。

- **git push:**

`git push` 用于将本地的提交推送到远程仓库。它执行以下操作：

- 将本地的提交上传到远程仓库。
- 更新远程仓库中的分支，以包含你的提交。

基本语法如下：`git push [远程仓库] [本地分支]:[远程分支]`

- `[远程仓库]` 是远程 Git 仓库的名称（通常是别名，如 `origin`）。
- `[本地分支]` 是你要推送的本地分支。
- `[远程分支]` 是你要将本地分支的提交推送到的远程分支。

例如，要将当前分支的更改推送到远程仓库的同名分支，你可以使用：`git push origin my-branch`

这将把 `my-branch` 的更改推送到 `origin` 远程仓库的 `my-branch` 分支上。

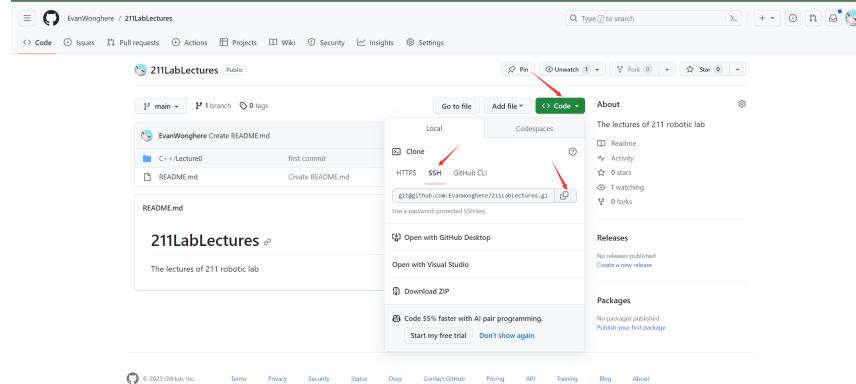
默认情况下，`git push` 命令将推送当前分支的更改到与之关联的远程分支。这意味着 Git 将尝试将你当前所在的分支的提交推送到远程仓库中同名的分支。

- **克隆仓库**

### 尝试将本课程的仓库克隆到本地

- 创建一个新的目录，并且在该目录打开 `Git Bash`。
- 克隆GitHub仓库

- GitHub仓库地址：<https://github.com/EvanWonghere/211LabLectures>
- 选择 `Code - Local - Clone - SSH` 然后复制（推荐使用SSH Clone而不是HTTPS Clone）。



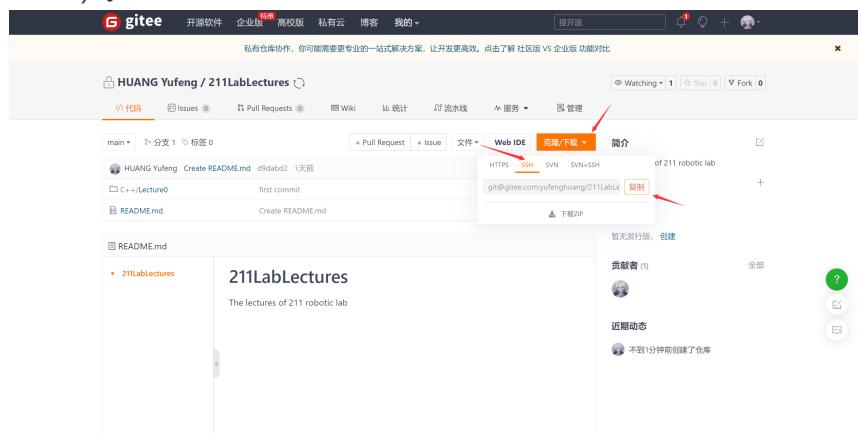
- 在 `Git Bash` 输入 `git clone`

`git@github.com:Evanwonghere/211LabLectures.git` 即可。

- 克隆Gitee仓库

- Gitee仓库地址：<https://gitee.com/yufenghuang/211LabLectures>

- 选择 克隆/下载 - SSH - 复制 (推荐使用SSH Clone而不是HTTPS Clone)。



- 在 Git Bash 输入 `git clone`

```
git@gitee.com:yufenghuang/211LabLectures.git 即可。
```