Git 基础（二）

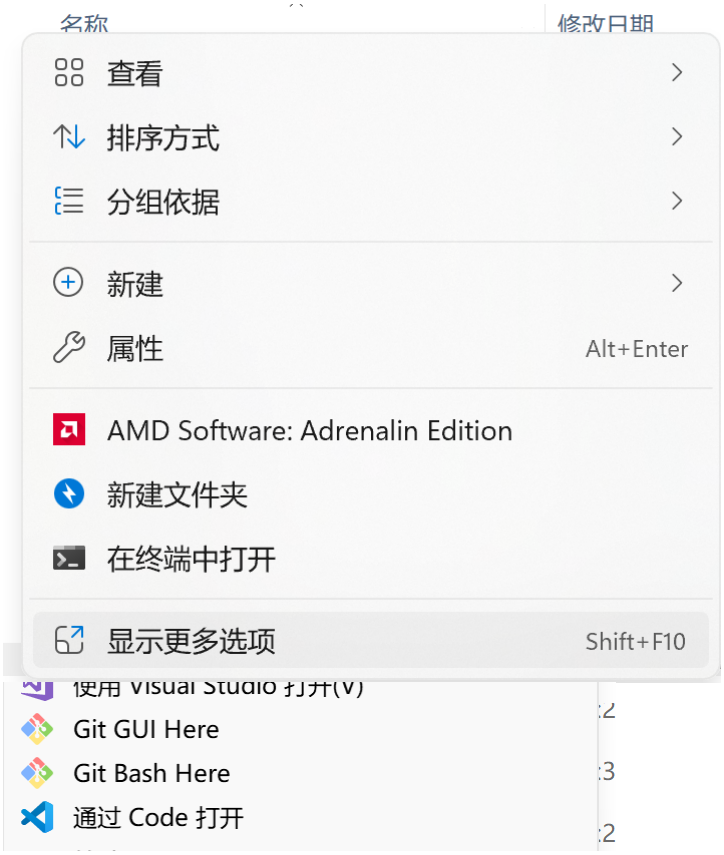
> 第一篇我们详细介绍了 Git 是干嘛的以及如何下载安装 Git，这一篇将会详细教大家如何玩转 Git，用 Git 完成许多不可思议的工作。

初始化 Git 工作区

- 在项目**根目录**下右键，

  选择**显示更多选项**，

  选中 `Git Bash Here` 一栏



> 如果没有这一栏？
>
> 检查是不是没有成功安装 Git？如果在命令行中输入 `git -v` 可以正常显示出版本号，则可能是系统注册表被修改了，这个时候你可以捧电脑过来我教你怎么重新把注册表写回去，但是更加推荐另外一种做法：
>
> - 在根目录下**右键**，直接选择**在终端中打开**，然后继续下面的步骤

- 打开 **Bash窗口** 或 **终端** 后，输入指令：

  ```
  git init
  ```

  即可初始化 Git 工作区。

  > 这是我们接触的第一条 Git 指令，Git 工具允许我们只用寥寥几条简单的指令来非常方便地版本控制，这一条指令的意思是初始化此文件夹的 Git 工作区，是第一条非常常用的、必须记住的指令。

.....

进行一系列文件/文件夹的增删改操作

.....

保存提交到本地仓库

在保存提交前，所有的操作都是基于暂存区进行的，所有的更改并没有被 **Track**，也就是你所做的东西并没有真正保存到仓库之中，如果你想在本地仓库生成一个版本快照时，你所需要做的第一件事就是**保存更改**，告诉 Git 你想通过 Git 来记录什么文件的增删改记录。我们可以只用一条非常简单的指令来告诉 Git 保存当前目录下所有文件的更改记录：

```
git add .
```

> 点号代表了我们希望全部文件都能被 Git 跟踪到更改，如果我们只想提交某个文件/文件夹，可以将点号改成所要添加的文件**相对路径**，如：
>
> ```
> git add /hello/hello.txt
> ```
>
> 则 Git 将会保存项目根目录下 `hello` 文件夹下的 `hello.txt` 文件的更改记录。

紧接着，我们就可以将所有已经保存添加的更改生成一个船新**版本**，也就是提交到**本地仓库**里，也只需要一行简单的指令：

```
git commit -m "版本描述信息"
```

执行这一指令后，Git 会根据我们所有通过 `git add` 添加的更改信息来生成一个**版本快照**，然后提交到**本地仓库**中，

我们可以通过 `git log` 指令来查看本地仓库的版本信息。

> `git commit` 指令必须要有后面的参数（`-m "xxx"`），`-m` 表示后面紧接着的一个参数是对这个版本的文字描述，将来推送到远程仓库时会显示在版本信息当中，除此之外，也可以不用 `-m` 而是 `-a`，这里先不介绍，在刚开始的大部分需求中，`-m` 就足以完成我们的大部分需求了。

目前为止，我们已经完成了**工作区初始化**、更改**保存提交**到**本地仓库**的工作了，我们可以回顾一下我们都干了什么：

- 一开始在项目根目录`git init`；
- 在项目根目录中进行任意更改...；
- `git add .`
- `git commit -m "描述一下版本信息"`

上面就是最简单的工作流程了，需要注意的是，每次想要提交到本地仓库前，一定要先提交更改，也就是先`git add`再`git commit -m "xxx"`

新建远程仓库

上面的操作都是在本地工作区或本地仓库来进行的，现在我们需要将本地仓库的版本推送到远程仓库中去。

首先我们需要在提供**Git远程仓库服务**的平台上注册一个账号，常见的有 **Github、Gitee**，

- Github：https://github.com
  - 完全免费并且使用限制极小
  - 网络环境不佳，有时会连接超时；
  - 需看懂一点英文
- Gitee：https://gitee.com
  - 国内平台，连接稳定
  - 在一定用量范围内是免费的，具体表现为一个账号**5个免费开源仓库**和无限的私有仓库，一个账号只能同时添加**5名以下**的好友作为协作者，而只有协作者才能访问你的私有仓库。
  - 使用受限，开源前需要审核。

那这里就以 **Gitee**，为例：

- 首先打开https://gitee.com，选择右上角**注册**，

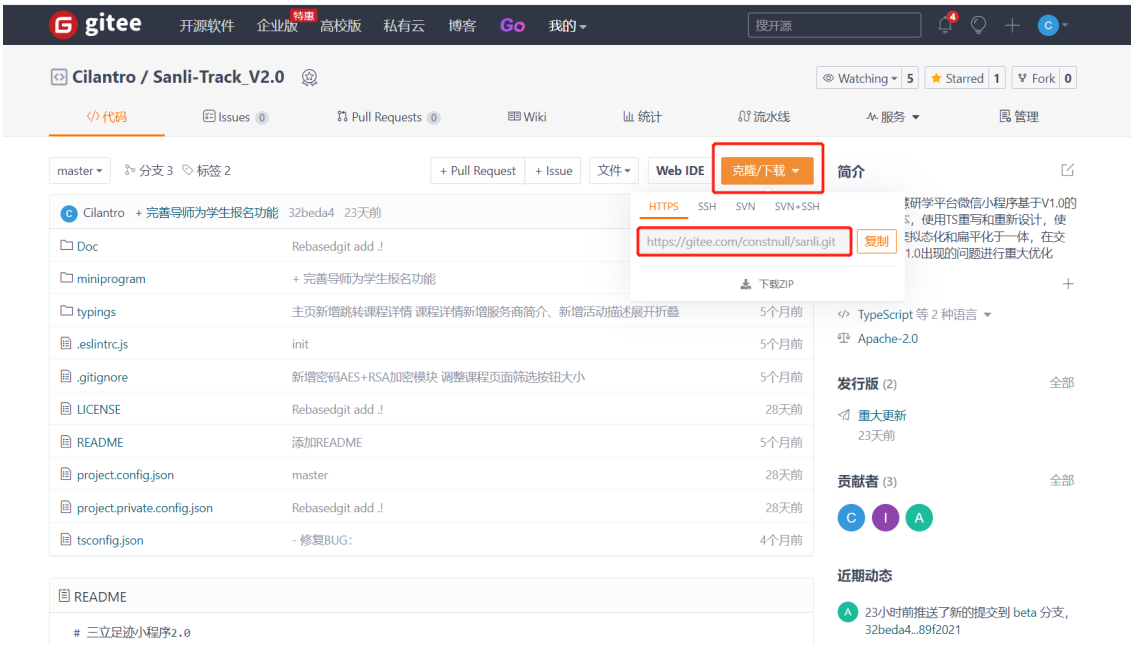
- 输入昵称、个人空间地址、手机号、密码。

> 个人空间地址是个人远程仓库的**唯一标识**，可以仔细起一个自己喜欢的名字（英文），以后也可以修改，但是修改前的**原仓库地址会失效**，意味着之前的本地仓库需要重新绑定远程仓库。

- 注册后直接登录即可。进入后，左下角可以看到自己的仓库，选择 `+` **创建仓库。**
- 进入新建仓库页面后：
  - 名称：随便起，会显示在Gitee个人界面中，相当于*名字*；
  - 路径：这个仓库在你的**个人空间**里的**唯一地址**，相当于*身份证*，只能是**全英文**；
  - 仓库介绍：随便填填；
  - 底下三个选项可以先不管，都不选；

> 新建仓库后，我们以后可以直接在浏览器中输入：`gitee.com + / + 个人空间地址 + / + 仓库路径` 来访问我们创建的远程仓库。

- 进入我们创建仓库之后，选择 **克隆/下载**，可以看到 **HTTPS** 选项中有一行地址，这就是我们**本地仓库绑定远程仓库**用的地址。直接把他**复制**下来。



配置 Git 个人信息

使用 Git 推送到远程仓库前，我们应该配置一下我们的个人信息，只需要 `Win + R` 打开并输入 `cmd` 后回车，在终端输入：

```
git config --global user.name "你的昵称"
git config --global user.email "你的邮箱地址"
```

这一个配置其实无关紧要，只是在后面推送到远程仓库时会作为记录推送来源的依据。

将本地仓库的版本推送到远程仓库

创建了自己的远程仓库，我们轻而易举拿到了**远程仓库的地址**（`https`开头，`.git`结尾的那个），这时候我们就可以拿他来给我们之前创建的本地仓库绑定了。

- 在刚才创建了 Git 工作区环境的目录，右键打开**终端**或`Git Bash`，输入：

  ```
  git remote add origin https://gitee.com/[个人空间地址]/[仓库地址].git
  ```

  我们来好好解析以下这一行是什么意思：

  - `origin`是我们**给远程仓库起的名字**，可以**随便起**，并不用和创建它的时候我们输入的仓库名字一样，只是我们在本地操作时用来**标识**某一远程仓库的**唯一名称**。
  - 后面紧跟的**URI**（统一资源标识符，也就是我们通常所说的网址），指示了我们要绑定的**远程仓库的地址**。
  - 一整句指令可以简单理解为，我们将`https://gitee.com/[个人空间地址]/[仓库地址].git`这一个远程仓库地址在本地起一个**别名**，叫作`origin`。

- 紧接着，假设我们已经**创建过本地仓库版本**（如果还没有请回头看前面），执行下面的指令：

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

  然后正常情况下，就会一片白的信息提示，最后有`success`字样，那这个时候我们就已经将本地仓库`master分支`的`HEAD`版本推送到远程仓库里的`master分支`了。

  > 在这里我们遇到了两个从没听过的词语：`master 分支`和`HEAD 版本`：
  >
  > - 分支，是 Git 的强大之处的又一体现。分支相当于仓库里面不同的货架，一个仓库里可以有非常多的"货架"，也就是非常多的分支，而`master分支`就是这个仓库里面的**默认分支**（主分支），在未经自定义的情况下，所有保存提交都是在`master分支`进行的，这里我们先不对**分支**这个模型有太多的了解，只需要知道`master分支`是一个仓库的**默认主分支**即可。
  > - 我们知道，版本是有唯一标识的版本号的（Git 为每一个版本偷偷起的名字），而处于本地仓库中的**最新版本**，有一个别名，就是`HEAD 版本`，我们使用 Git 时的推送/拉取操作有非常多都是基于 **HEAD 版本** 进行的。

  解释完一些必须知道的名词之后，我们回到这一句指令看一看它到底干了什么：

  - `git push`表示推送本地仓库中的版本到远程仓库；
  - `-u`是`--set-upstream-to`的简写，也就是指示 **绑定** 这一操作；

- `origin` 是我们刚才起的远程仓库别名；
- `master` 是指示我们希望推送到的远程仓库的**分支**。

> - 我们只用在第一次推送操作时使用 `-u`，也就是只用绑定一次远程仓库的分支，其他时候我们只需要：
>
>   ```
>   git push
>   ```
>
>   就可以完成推送操作（前提是我们已经在**第一次推送时**使用 `-u` 来**绑定本地分支和远程分支**）
> - 当然我们也可以为每次推送指定推送到的远程分支，只需要删掉 `-u`，然后照常加上【远程仓库别名】和【远程分支】即可，例如：`git push origin beta` 表示推送到 `origin` 远程仓库的 `beta` 分支。

目前为止，我们终于把我们的第一个本地仓库版本推送到了远程仓库版本，日后继续进行增删改操作后，我们可以再次重复上面的流程（只不过不用再绑定远程仓库了）：

- `git add .` 保存更改信息；
- `git commit -m "xxx"` 提交一个新版本到本地仓库；
- `git push` 推送到远程仓库。
- ……

而从远程仓库**拉取**的操作也非常简单，如果你之前使用`git push -u`来绑定过远程仓库的分支，则只需要执行：

```
git pull
```

就可以将对应远程分支的 **HEAD版本** 拉取到本地仓库，并且成为**本地的HEAD**

如果之前没有绑定远程仓库分支，则需要稍作修改，改为：

```
git pull [远程仓库别名] [远程分支]
```

上面就是最简单的使用 Git 来实现版本控制的例子了，那么更加进阶的 Git 操作，比如 **checkout、branch、rebase、merge/fetch** 等等我们会在后面使用到的时候会再详细介绍。

现在罗列一些除了上面介绍的以外的常用的 Git 指令：

```
# 克隆远程仓库（本地可以不预先初始化 Git 工作区，将会将远程仓库的 HEAD 版本直接复制到本地并创建本地仓库）
git clone [远程仓库地址]
# 删除远程仓库的绑定信息
git remote rm [远程仓库别名]
# 新建分支
git checkout -b [新分支名称]
# 切换分支
git checkout [分支名]
# 合并分支
```

```
git merge [要合并进来的本地分支名]
# 在其他分支基础上重新建立基底
git rebase [基底分支名]
# 查看当前 Git 配置
git config -l
# 查看 Git 全局配置
git config --global -l
```