Git指南

通过本文档你将学到：

一、认识Git

最开始接触Git，可能之前连听说都没听说过。我也是第一次接触，以下内容纯属现学现卖，我肯定会以通俗的方式讲解Git

1.1 为什么用Git

如果你写过长篇大论，那你一定有这样的经历：

想删除一个段落，又怕将来想恢复找不回来怎么办？有办法，先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去。

过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找

有些部分需要你同事帮助填写，于是你把文件Copy到U盘里给她（也可能通过Email发送一份给她），然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，**发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并**

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？

这个软件用起来就应该像这个样子，能记录每次文件的改动：

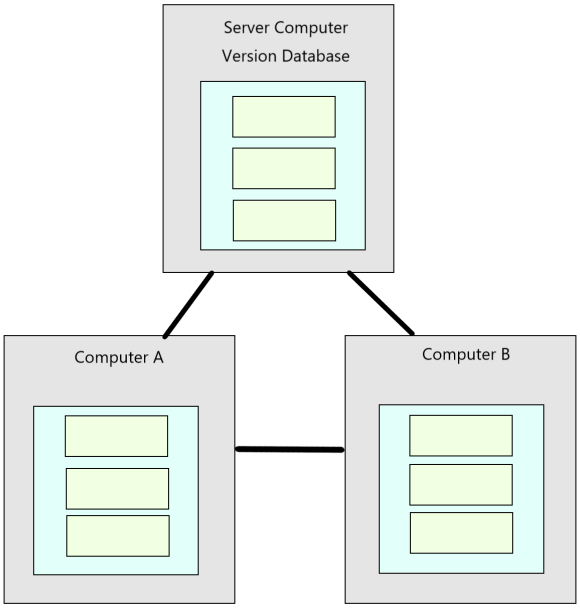
版本	文件名	用户	说明	日期
1	service.doc	张三	删除了软件服务条款5	7/12 10:38
2	service.doc	张三	增加了人数限制	7/12 18:09
3	service.doc	李四	调整了合同金额	7/13 9:51
4	service.doc	张山	延长了免费升级周期	7/14 15:17

所以git便是你的不二之选

1.2 什么是Git

Git是目前世界上最先进的**分布式版本控制系统**

分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就**不需要联网**了，因为**版本库就在你自己的电脑上。**既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，**你们俩之间只需把各自的修改推送给对方**，就可以互相看到对方的修改了



在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也**有一台充当“中央服务器”的电脑**，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。

二、配置Git

2.1 安装，安装后配置请见git初始化配置

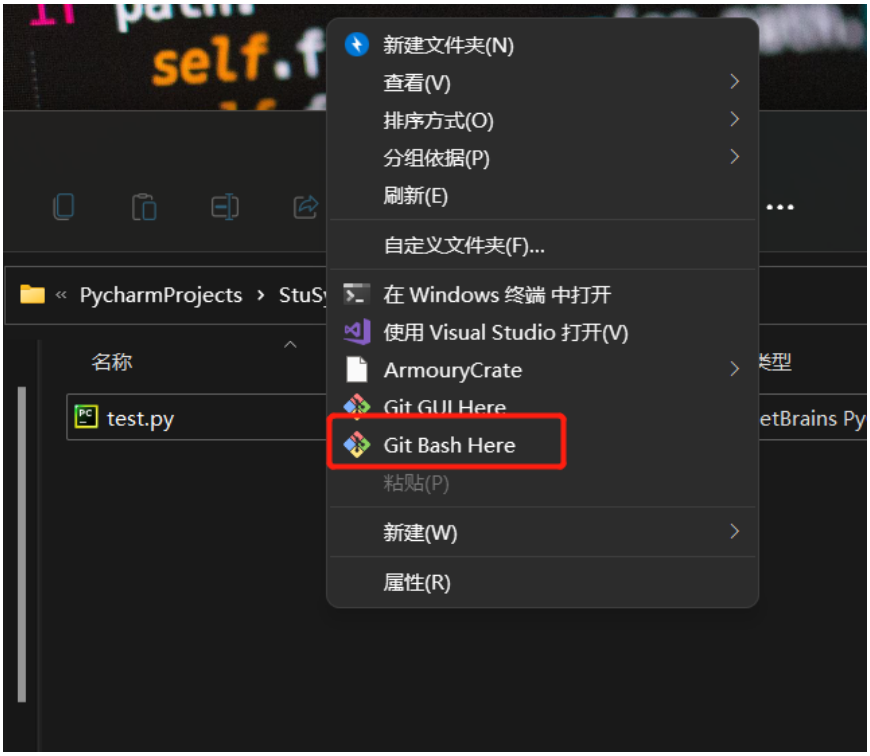
2.2 配置版本库

版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

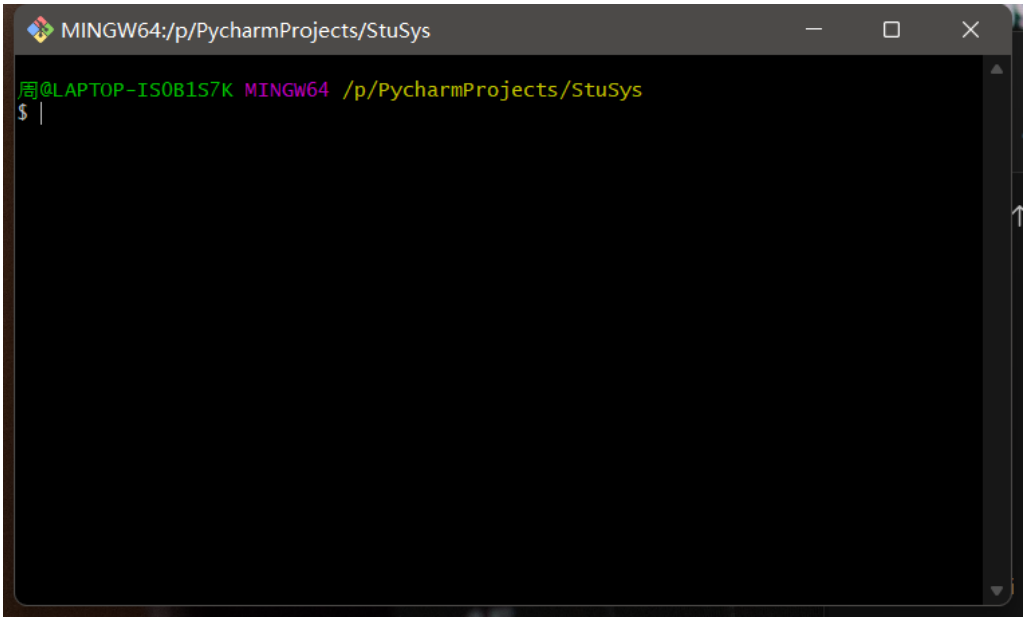
2.2.1 创建版本库文件夹

#注：如果你使用*Windows*系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

打开某一**文件夹**，右击鼠标，点击**Git Bash Here**



这时就会弹出一个小窗口



输入通过`git init`命令把这个目录变成Git可以管理的仓库

```
$ git init
Initialized empty Git repository in P:/PycharmProject/StuSys/.git
```

如果是空文件夹会告诉你这是一个空的仓库（empty Git repository）

之后我们就会看见这个文件夹下自动生成了一个.git文件夹

这个文件夹默认是隐藏的，看不看见无所谓，这个目录是Git来跟踪管理版本库的，**没事千万不要手动修改这个目录里面的文件**，不然改乱了，就把Git仓库给破坏了

2.2.2 把文件添加到版本库

> 所有的版本控制系统，其实只能跟踪文本文件的改动，比如**TXT文件，网页，所有的程序代码**等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词，在第8行删了一个单词。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道
>
> 不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，如果要真正使用版本控制系统，就要**以纯文本方式**编写文件
>
> 因为文本是有编码的，比如中文有常用的GBK编码，日文有Shift_JIS编码，如果没有历史遗留问题，强烈建议使用标准的**UTF-8编码**，所有语言使用同一种编码，既没有冲突，又被所有平台所支持

建议下载Visual Studio Code代替记事本

行，我们言归正传

这里我们新建一个helloworld.py文件，写入以下代码

```
print("helloworld")
```

一定要放到目录下，因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件

第一步、在小窗口执行`git add`添加文件

```
$ git add helloworld.py
```

没有任何提示，但是恰恰说明了已经成功添加到git

第二步、用`git commit`指令，把文件提交到仓库

```
$ git commit -m "new a helloworld python  file"
[master (root-commit) 8520036] new a helloworld python file
1 files changed, 1 insertion(+)
creat mode 100644 helloworld.py
```

`git commit`命令执行成功后会告诉你，`1 file changed`：1个文件被改动（新添加的helloworld.py文件）；`1 insertions`：插入了一行内容（helloworld.py有一行内容）

为什么Git添加文件需要`add`，`commit`一共两步呢？因为`commit`可以一次提交很多文件，所以你可以多次`add`不同的文件，比如：

```
$ git add file1.txt
$ git add file2.txt file3.txt
$ git commit -m "add 3 files."
```

添加文件到Git仓库，分两步：

1. 使用命令`git add <file>`，注意，可反复多次使用，添加多个文件；
2. 使用命令`git commit -m <message>`，完成

2.2.3 查询

如何查询我们当前`add`多少文件呢

比如：我们在目录下新建两个文本文件：subarashi.txt, sugoi.txt 然后输入`add`指令

```
$ git add sugoi.txt
```

之后我们再输入`git status`指令

```
$ git status
On branch master
Change to be committed:
  (use"git retore --staged<file>..." to unstage)
       new file:  sugoi.txt

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

此处`new file:  sugoi.txt`为绿色字体，`subarashi.txt`为红色字体，则表示sugoi.txt文件为已经`add`，准备`commit`；而subarashi.txt并未`add`。从中我们也知道，使用`git retore --stage<flie>`也可以撤回准备`commit`的文件

至于已经commit的文件，就不会再显示，如果当前没有add文件，则会显示

```
nothing added to commit but untrack files present (use "git add" to track)
```