

Git教程

以下内容基于Git教程 - 廖雪峰的官方网站 (liaoxuefeng.com), 对于Git用法进行相应的整理总结。仅用于个人向学习 Git。

—Yifan Jiang

Git安装

Linux系统上安装Git

此处以我目前的 Linux 操作系统 Ubuntu22.04.3-LTS 为例, 我们在 bash 中输入 `git` :

```
$ git
The program 'git' is currently not installed. You can install it by typing:
sudo apt-get install git
```

根据上述信息, 如果没有安装 `git`, 我们可以使用 `sudo apt-get install git` 语句直接安装。

安装完成之后输入 `git` 就可以得到如下所示的提示:

```
jyf@DESKTOP-ANMD07D: $ git
usage: git [--version] [--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.
jyf@DESKTOP-ANMD07D: $
```

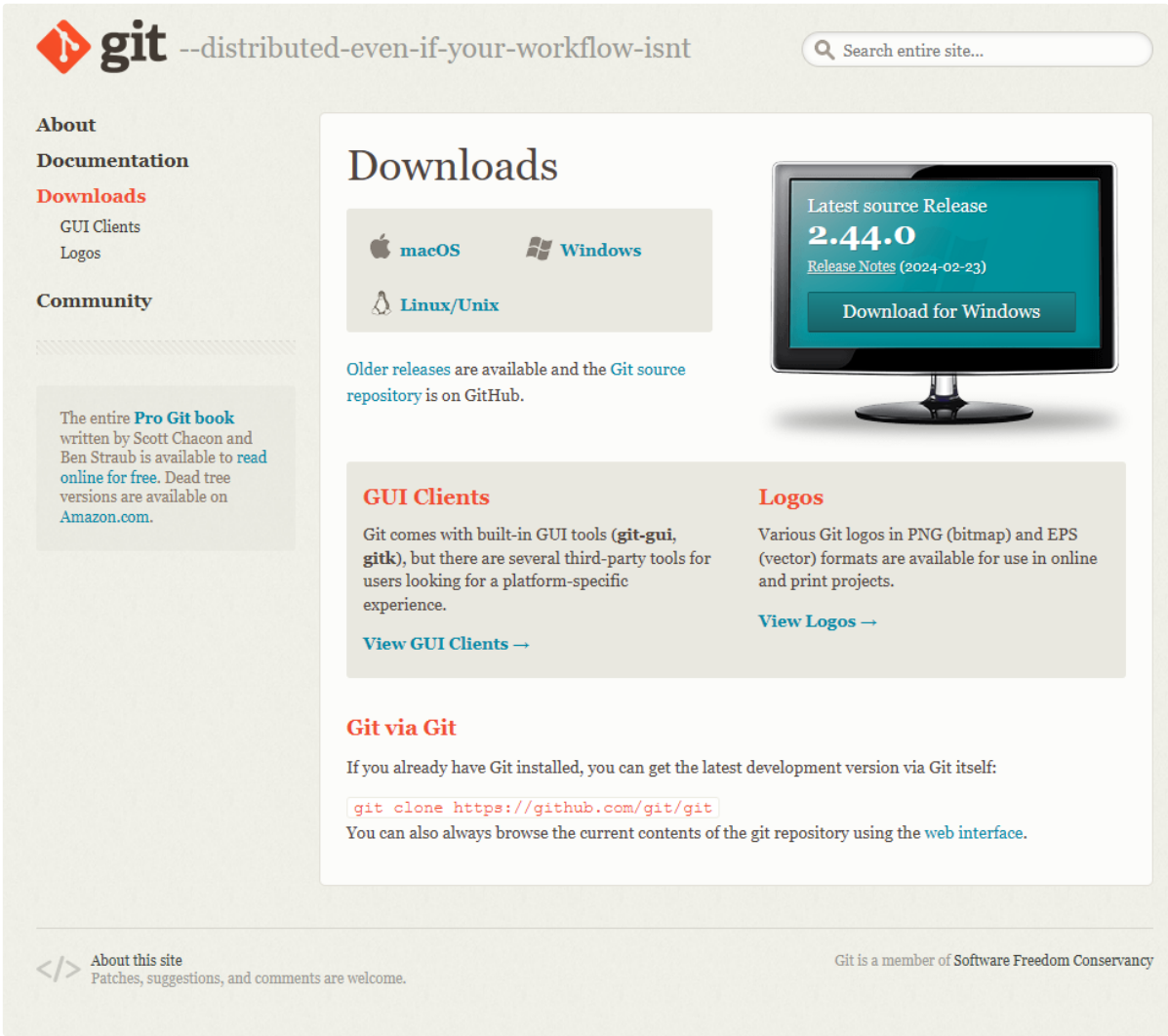
Mac OS上安装Git

1. 通过安装 `homebrew` 安装git。
2. 通过 AppStore 中的 Xcode 进行安装。

不过这边没用过Mac, 可能提供不了其他帮助 ((

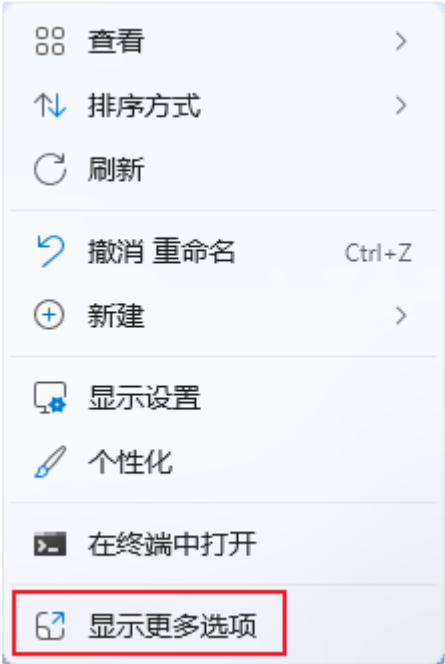
Windows系统上安装Git

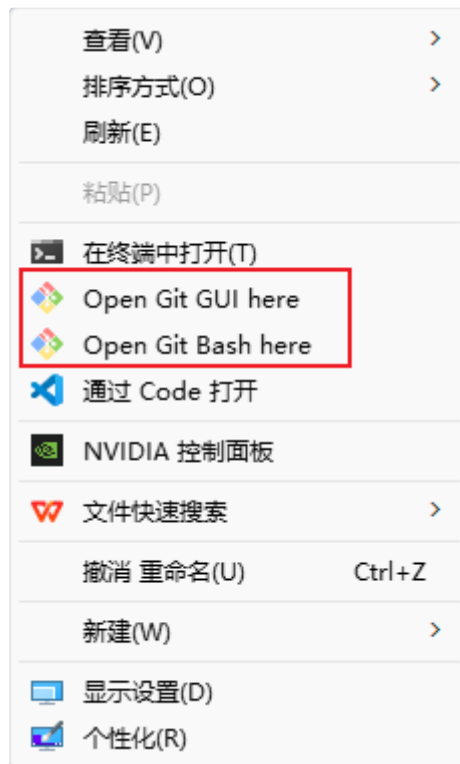
可以直接上Git官网 [Git - Downloads \(git-scm.com\)](https://git-scm.com) 进行安装。



Git基础设置

安装完成之后，我们在桌面的空白处右键鼠标，可以发现多出来两个选项（Win11的用户需要先点击“显示更多选项”）：





我们这里点击 **Open Git Bash here** 并且需要设置一下基础设置，即为你的电脑取一个名字，并且留一个 email。

在命令行中输入：

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

由于 Git 是分布式版本控制系统，因此每一个机器都需要有一个身份标识，在每次提交或者修改内容时候可以让其他开发者知道是谁操作的。

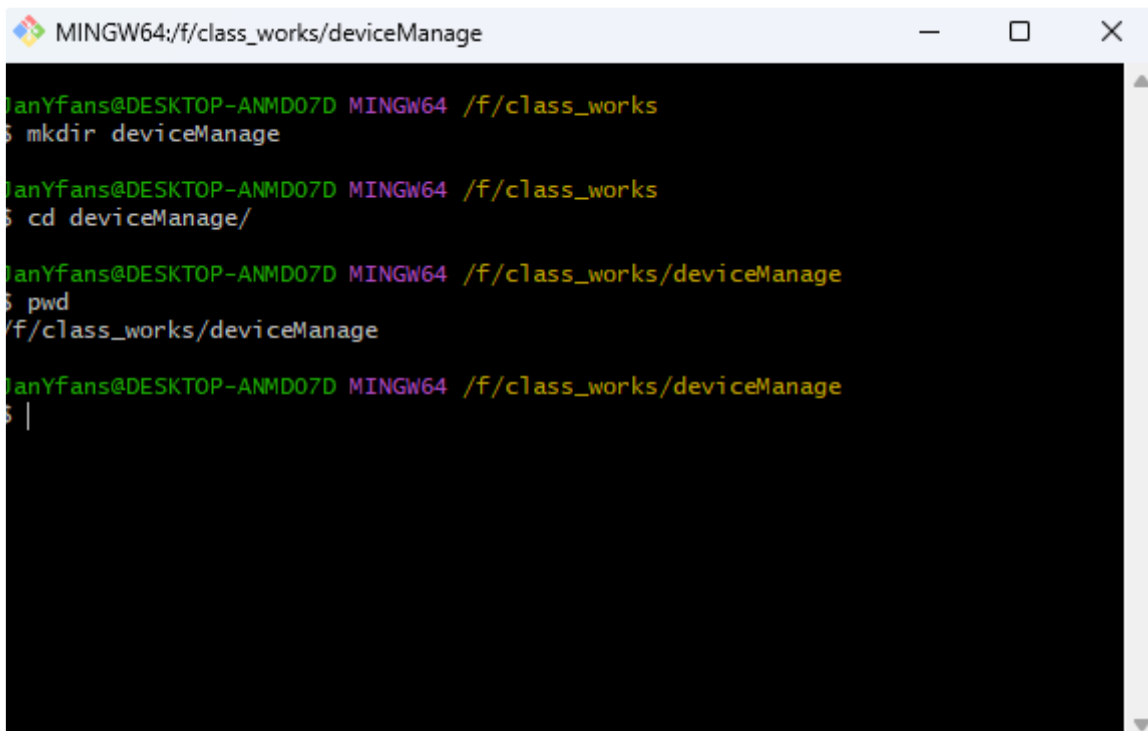
版本库的创建与使用

版本库，又名仓库(repository，简称repo)。在本地电脑上可以理解为一个目录，对这个目录当中的文件进行修改、删除等操作都可以被 Git 进行追踪。也就是说只要你想，你可以将这个目录内的文件回退到任何时候的版本。

创建版本库

新建一个空文件夹，并在这个文件夹内鼠标右键，选择 **Open Git Bash here** 然后输入在 bash 中输入以下命令：

```
$ mkdir deviceManage
$ cd deviceManage
$ pwd
/f/class_works/deviceManage
```



```
MINGW64:/f/class_works/deviceManage
JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works
$ mkdir deviceManage

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works
$ cd deviceManage/

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ pwd
/f/class_works/deviceManage

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ |
```

其中,

- `mkdir deviceManage` 命令是在当前目录下创建一个名为 `deviceManage` 的文件夹, 这个名字可以随便起。
- `cd deviceManage` 命令是进入 `deviceManage` 文件夹, 同样, 如果该目录下有其他文件夹, 则这个名字可以替换。
- `pwd` 命令是显示当前目录, 例如我的当前目录是 `/f/class_works/deviceManage`, 也就是 `bash` 中黄色字体显示的内容。

Important

▲ `bash` 中的复制粘贴不是 Windows 系统下的 `Ctrl+C` 和 `Ctrl+V`, 作为个人来说一般不使用快捷键 (因为记不住), 而是直接右键选择相应的操作。

```
JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works
$ mkdir deviceManage

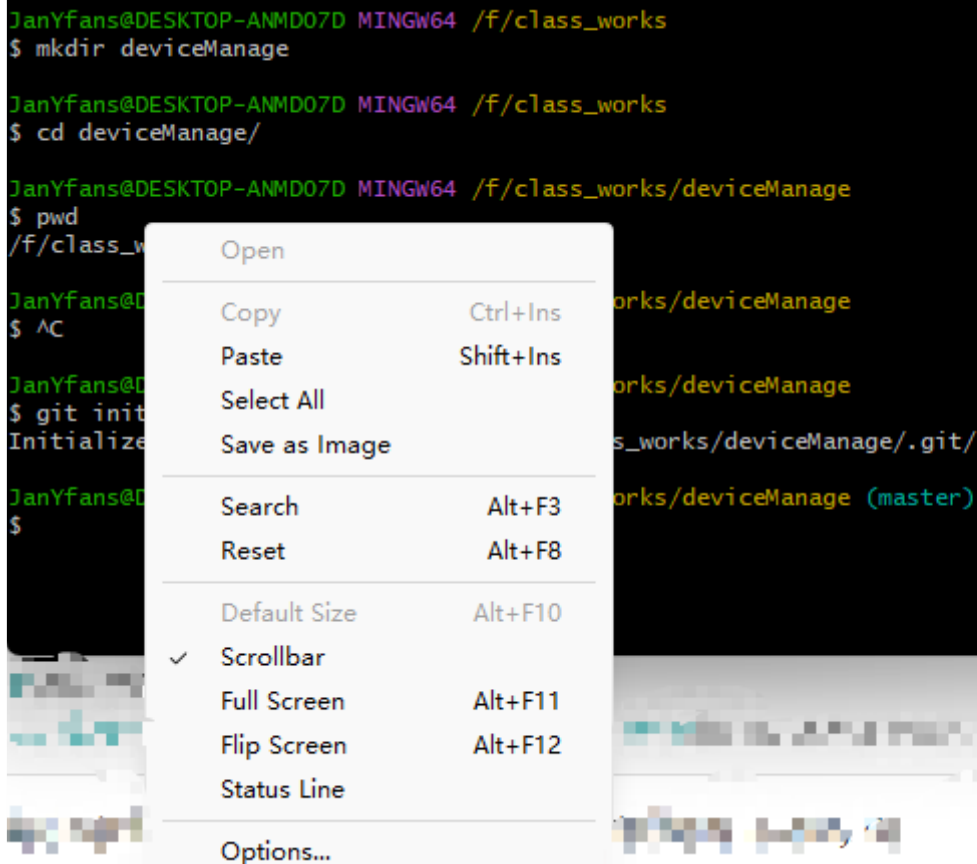
JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works
$ cd deviceManage/

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ pwd
/f/class_w

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ AC

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ git init
Initialized empty Git repository in F:/class_works/deviceManage/.git/

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage (master)
$
```



这时我们来到了一个空文件夹中，于是我们可以开始 Git 仓库的构建了。

我们使用 `git init` 命令进行仓库的创建：

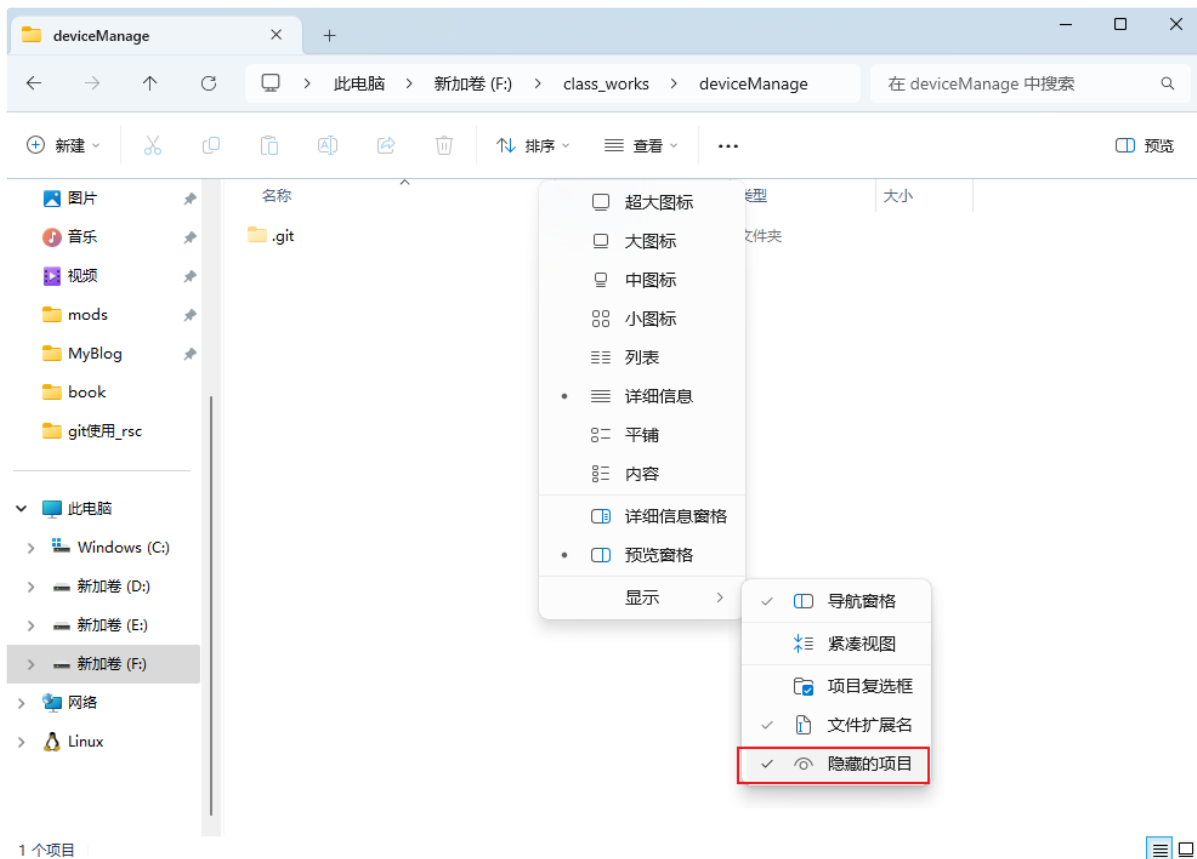
```
$ git init
Initialized empty Git repository in F:/class_works/deviceManage/.git/
```

```
JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage
$ git init
Initialized empty Git repository in F:/class_works/deviceManage/.git/

JanYfans@DESKTOP-ANMD07D MINGW64 /f/class_works/deviceManage (master)
$
```

在输入完成之后你可以发现多出了 `(master)`，这表明这个目录是一个 Git 仓库。

同时，如果打开了 Windows 的“隐藏的项目”的设置，我们可以看到这个目录中有一个 `.git` 的文件夹，这个文件夹是默认隐藏的，Git 就是**使用这个文件夹进行版本库管理的**。如果删除或者修改了这个目录里面的文件，很有可能会将 Git 仓库破坏，导致项目的损坏。



使用版本库

将文件添加到版本库

由于所有的版本控制软件只能跟踪**文本文件**（例如TXT、HTML、程序代码等）的改动，因此对于图片、视频等**二进制文件**，Git 是无法获取到底更改了啥东西，只知道文件大小改变了（例如图片的大小从100KB→120KB）。

同时，由于 Microsoft 的 Word 是二进制格式，因此无法跟踪 Word 类文件的改动。

⚠ Caution

❓ 为什么不推荐使用 Windows 的记事本编写**任何**文本文件：

- 效率较低
- Microsoft的记事本在保存 UTF-8 编码的文件时，会自动在每个文件开头添加一个 `0xEFBBBF` 的字符，导致在运行程序或者提交代码的时候出现乱码或者报错。

因此一般推荐使用功能强大的 **Visual Studio Code**（简称 VS Code）代替记事本。

安装好 VS Code 之后，我们在当前目录鼠标右键可以发现多出了一个选项（我这边是早就装好了）：



之后便可以很方便地进行创建文件，修改文件等操作啦。

回到正题，现在在我们的**工作目录(master)**下创建一个 `readme.txt` 文件，然后随便写点内容进去，例如：

```
Hello Git!  
Hello VS Code!
```

1. 我们在 `bash` 窗口中输入以下命令，告诉 `Git` 我们有一个新的文件**添加到仓库**：

```
$ git add readme.txt
```

如果没有任何显示，就说明没有任何问题。

2. 我们输入 `git commit` 命令，将 `readme.txt` **提交到仓库**：

```
$ git commit -m "wrote a readme file"  
[master (root-commit) 9f97037] wrote a readme file  
1 file changed, 2 insertions(+)  
create mode 100644 readme.txt
```

其中：

- `git commit` 命令后加上 `-m "任意内容"` 代表这次提交的**注释**，在开发过程中**尤为重要**，因为这样可以在之后查看历史记录的时候了解这次提交了啥东西。
- 在输入 `git commit` 命令之后，出现的 `1 file changed` 代表 1 个文件被改动；`2 insertions(+)` 代表插入了 2 行内容。

git add 和 git commit 的区别

`git add` 一次性是添加一个文件, `git commit` 一次性可以提交多个文件。

可以理解为 `add` 是将文件在进入仓库前一个个排好队, 然后 `commit` 就类似于将仓库门打开让这个队伍里的所有人进入仓库。

所以可以多次 `add` 不同的文件, 然后一次性 `commit` 上去, 例如:

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

* 命令

git add

→ 作用: 将单/多个文件添加到暂存区。

→ 语法:

- `git add [file1] [file2] ...` : 将单/多个文件添加到暂存区
- `git add [dir]` : 将指定目录添加到暂存区(包括子目录)
- `git add` : 将当前目录下的**所有文件**添加到暂存区

💡 Tip

PS: 若要**查看详细语法**, 请在 `bash` 中输入 `git -h add` 进行查看。

git commit

→ 作用: 将多个文件提交到本地仓库。

→ 语法:

- `git commit -m [message]` : 提交暂存区到本地仓库中
- `git commit [file1] [file2] ... -m [message]` : 提交指定文件到本地仓库
- `git commit -am [message]` : 修改文件后**无需使用** `git add`, 直接提交到本地仓库

📌 Note

设置提交代码时的用户信息

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

如果去掉 `--global` 参数则只对当前仓库有效

💡 Tip

PS: 若要**查看详细语法**, 请在 `bash` 中输入 `git -h commit` 进行查看。

使用 Git 进行版本管理

仓库状态

我们将 `readme.txt` 文件改成以下内容并保存：

```
Git Hello!  
VS Code Hello!
```

然后运行 `git status` 命令查看结果：

```
$ git status  
On branch master  
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.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

- 使用 `git status` 命令可以时刻掌握仓库当前的状态，可以看到有一个 `modified: readme.txt` 说明 `readme.txt` 被修改过了，但是还未提交到仓库。

如果想要查看具体修改了哪些内容，我们可以使用 `git diff` 命令进行查看：

```
$ git diff readme.txt  
diff --git a/readme.txt b/readme.txt  
index 9b3120e..0db40c6 100644  
--- a/readme.txt  
+++ b/readme.txt  
@@ -1,2 +1,2 @@  
-Hello Git!  
-Hello VS Code!  
\ No newline at end of file  
+Git Hello!^M  
+VS Code Hello!  
\ No newline at end of file
```

在 `@@ -1,2 +1,2 @@` 中我们可以看到增加和减少了那些内容，具体内容是这个语句下方 `-` 和 `+` 开头的语句。例如：`-Hello Git!` 代表我们将这个内容删除了。

然后我们使用 `git add` 进行添加文件：

```
$ git add readme.txt  
$ git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
        modified:   readme.txt
```

- 使用 `git status` 告诉我们，将要被提交的修改(`Changes to be committed`)包括 `readme.txt` 。没有报错，因此我们可以直接进行提交了：

```
$ git commit -m "modify readme"
[master d6032ba] modify readme
1 file changed, 2 insertions(+), 2 deletions(-)
```

提交完成后，我们再次使用 `git status` 命令查看当前仓库状态：

```
$ git status
On branch master
nothing to commit, working tree clean
```

此时 Git 告诉我们当前没有需要提交的修改，此时工作目录是干净的（`nothing to commit, working tree clean`）。

版本回退

现在，我们学会了修改文件，并把这个修改提交到 Git。再次修改readme.txt文件如下：

```
Git Hello 11111111!
VS Code Hello!
```

然后尝试提交：

```
$ git add readme.txt
$ git commit -m "add 11111111"
[master 377e979] add 11111111
1 file changed, 2 insertions(+), 2 deletions(-)
```

就像存档一样，在打BOSS之前为了保险起见都会先存个档，如果某一关没过去，可以选择读取前一关的状态。每当你觉得文件修改到一定的程度，可以“存一个档”，一旦文件被误删或者改乱了，可以“读档”将文件回复（以防删库跑路）。

现在我们回忆一下 `readme.txt` 一共有几个版本被提交到 Git 仓库中了：

Version1: wrote a readme file

```
Hello Git!
Hello VS Code!
```

Version2: modify readme

```
Git Hello!
VS Code Hello!
```

Version3: add 11111111

```
Git Hello 11111111!
VS Code Hello!
```

对于查看历史记录，我们可以通过使用 `git log` 进行查看：

```
$ git log
commit 377e979b4b7c8fa0189d1fd7b8ff8ad6ef8cad87 (HEAD → master)
Author: Nuame <623857032@qq.com>
Date: Sun Mar 17 11:25:51 2024 +0800

    add 11111111
```

```
commit d6032bae9953fa7c6dbd1e4732bf1af011a0d6fe
Author: Nuame <623857032@qq.com>
Date: Sun Mar 17 11:13:37 2024 +0800
```

modify readme

```
commit 9f9703727f6e633e25207751731b194254d73b5c
Author: Nuame <623857032@qq.com>
Date: Sun Mar 17 01:45:33 2024 +0800
```

wrote a readme file

可以看到，上面的 **Author** 部分就是我们在Git基础设置内设置的 **name** 和 **email** 属性。

然后，我们发现 **git log** 命令显示的是从最近到最远的提交日志，总共有三次提交，其中最近的一次是 **add 11111111**，上次是 **modify readme**，最早一次是 **wrote a readme file**。

如果觉得信息太多，可以使用 **git log --pretty=oneline** 进行查看：

```
$ git log --pretty=oneline
377e979b4b7c8fa0189d1fd7b8ff8ad6ef8cad87 (HEAD -> master) add 11111111
d6032bae9953fa7c6dbd1e4732bf1af011a0d6fe modify readme
9f9703727f6e633e25207751731b194254d73b5c wrote a readme file
```

④ Note

注意到，上述详细日志和简化日志都含有类似于 **377e9...cad87** 的东西，这个是 **commit id**，就像我们每个人都会有一个身份证号，用于区别你我他。Git 的 **commit id** 是通过SHA-1计算出来的一个大数字，用十六进制表示。

由于是多人进行协作更新，如果使用 **1, 2, 3 ...** 作为版本号，肯定会产生冲突。

每提交一个新版本，Git 会自动将它们串联成一条时间线。

如果我们想要将目前的 **add 11111111** 版本回退到 **modify readme** 版本，那要如何去做呢？

首先，Git 必须知道是哪个版本，在 Git 中，使用 **HEAD** 表示当前版本，例如我的电脑上是

377e979... 为当前的最新提交。

- 上一个版本可以使用 **HEAD^**，上上一个版本可以使用 **HEAD^^**。
- 如果回退的版本比较多，例如回退100个版本，可以使用 **HEAD~100**（当然你写100个 **^** 也是可以的）。

使用 **git reset** 命令进行版本回退：

```
$ git reset --hard HEAD^
HEAD is now at d6032ba modify readme
```

然后我们查看 **readme.txt** 的内容，看看是不是之前的版本：

```
$ cat readme.txt
Git Hello!
VS Code Hello!
```

正好是Version2的版本。

我们可以继续回退到**上一个版本**（即Version1），但是如果让我们使用 **git log** 查看当前版本库状态：

```
$ git log
commit d6032bae9953fa7c6dbd1e4732bf1af011a0d6fe (HEAD → master)
Author: Nuame <623857032@qq.com>
Date: Sun Mar 17 11:13:37 2024 +0800

    modify readme

commit 9f9703727f6e633e25207751731b194254d73b5c
Author: Nuame <623857032@qq.com>
Date: Sun Mar 17 01:45:33 2024 +0800

    wrote a readme file
```

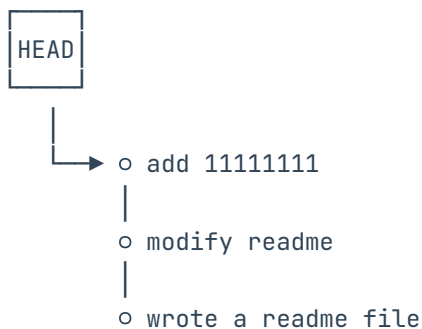
发现Version3的版本不见了!! 如果我们没有新开一个命令行窗口的话, 可以通过查找上面的Version3的 `commit id` 进行回档。

```
$ git reset --hard 377e9
HEAD is now at 377e979 add 11111111
```

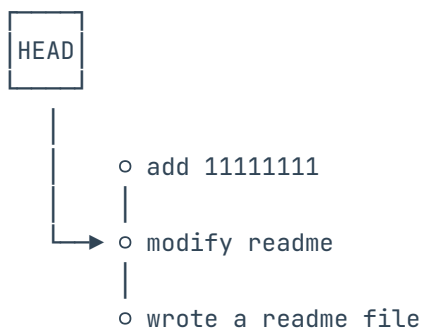
再次查看 `readme.txt` 的内容:

```
$ cat readme.txt
Git Hello 11111111!
VS Code Hello!
```

Git 的版本回退速度非常快, 因为 Git 内部有一个指向当前版本 `HEAD` 的指针, 当回退版本的时候, 只需要更改指针指向的内容即可:



改为指向 `modify readme` :



然后将工作区的文件更新了。

▲如果想恢复到新版本, 但是找不到之前的 `commit id` 了怎么办? 可以使用 `git reflog` 查看命令历史。

```
$ git reflog
377e979 (HEAD → master) HEAD@{0}: reset: moving to 377e9
d6032ba HEAD@{1}: reset: moving to HEAD^
377e979 (HEAD → master) HEAD@{2}: commit: add 11111111
d6032ba HEAD@{3}: commit: modify readme
9f97037 HEAD@{4}: commit (initial): wrote a readme file
```

通过查看历史记录我们可以发现 `add 11111111` 的 `commit id` 是 `377e979`。

* 命令

git status

→ 作用：用于查看 Git 仓库当前状态

→ 语法：

- `git status` : 查看上次提交之后是否有对文件再次修改
- `git status -s` : 获得简短的输出结果

💡 Tip

PS：若要查看详细语法，请在 `bash` 中输入 `git -h status` 进行查看。

git diff

→ 作用：比较文件在暂存区和工作区的差异

→ 语法：

- `git diff [file]` : 查看暂存区和工作区的差异
- `git diff --cached [file]` : 显示暂存区和上一次提交 `commit` 的差异
- `git diff --staged [file]` : 显示暂存区和上一次提交 `commit` 的差异
- `git diff [first-branch]...[second-branch]` : 显示两次提交之间的差异

💡 Tip

PS：若要查看详细语法，请在 `bash` 中输入 `git -h diff` 进行查看。

git reset

→ 作用：切换不同版本

→ 语法：

- `git reset [--soft | --mixed | --hard] [HEAD]`
 - `--mixed` 为默认，可以不带该参数，用于重置暂存区的文件与上一次提交 `commit` 保持一致，工作区文件内容不变。
 - `--soft` 参数用于回退到某个版本。
 - `--hard` 参数用于撤销工作区中的所有未提交的修改内容，将暂存区和工作区都回到上一次版本。
- `HEAD / HEAD~0` : 表示当前版本
- `HEAD^ / HEAD~1` : 表示上一个版本

- HEAD^^ / HEAD~2 : 表示上上一个版本

💡 Tip

PS: 若要**查看详细语法**, 请在 bash 中输入 `git -h reset` 进行查看。

git log

→ 作用: 查看**提交历史**。

→ 语法:

- git log [选项] [分支名/提交哈希]

📌 Note

常用的选项包括:

- `-p` : 显示提交的补丁
- `--oneline` : 以简洁的一行格式显示提交信息
- `--graph` : 以图形化方式显示分支和合并历史
- `--decorate` : 显示分支和标签指向的提交
- `--no-merges` : 不显示合并提交
- `--stat` : 显示简略统计信息, 包括修改的文件和行数
- `--abbrev-commit` : 使用短提交哈希值
- `--author=<作者>` : 只显示特定作者的提交
- `--since=<时间>` : 只显示指定时间之后的提交
- `--until=<时间>` : 只显示指定时间之前的提交
- `--graph=<模式>` : 只显示包含指定模式的提交消息
- `--pretty=<格式>` : 使用自定义的提交信息显示格式

💡 Tip

PS: 若要**查看详细语法**, 请在 bash 中输入 `git -h log` 进行查看。

git reflog

→ 作用: 查看命令历史

→ 语法:

- git reflog

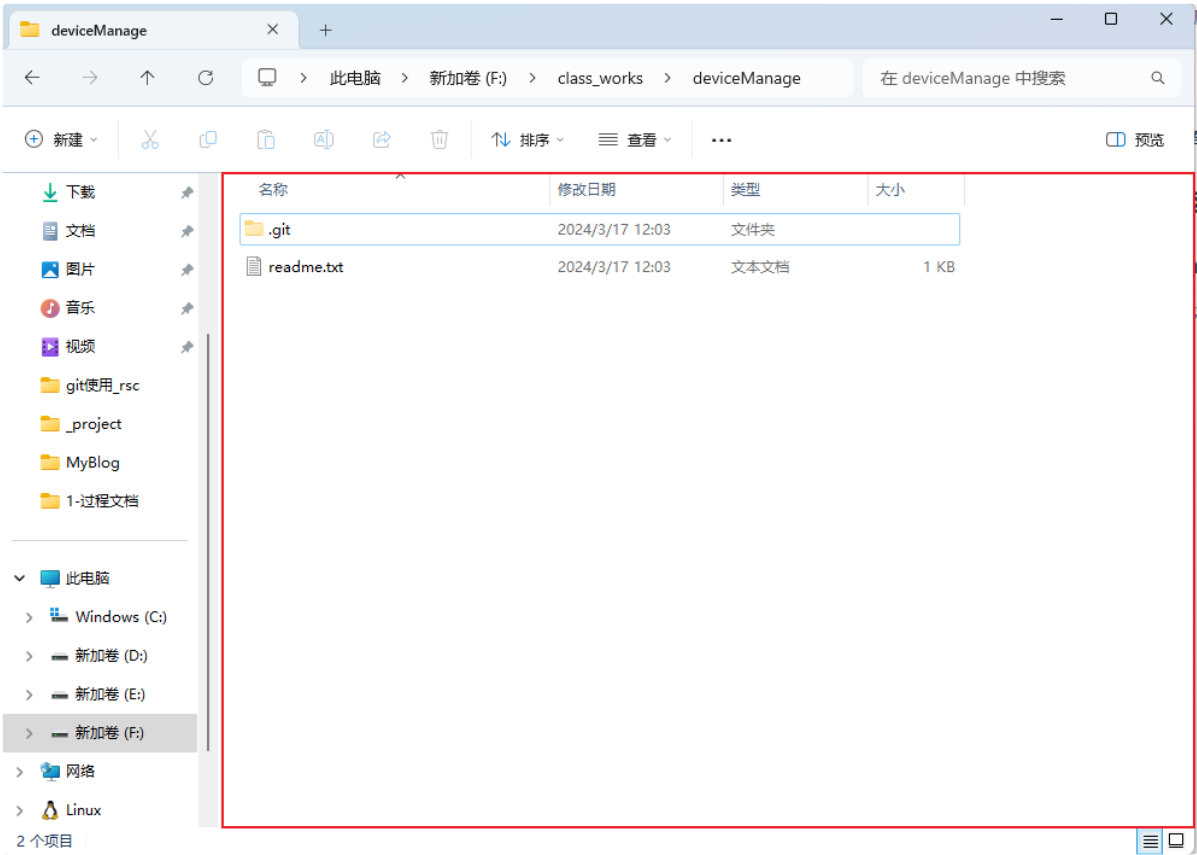
💡 Tip

PS: 若要**查看详细语法**, 请在 bash 中输入 `git -h reflog` 进行查看。

工作区和暂存区

工作区 (Working Directory)

实际上应该被翻译成“**工作目录**”，就是 `.git` 所在的文件目录中，例如：



就是一个 Git 工作区。

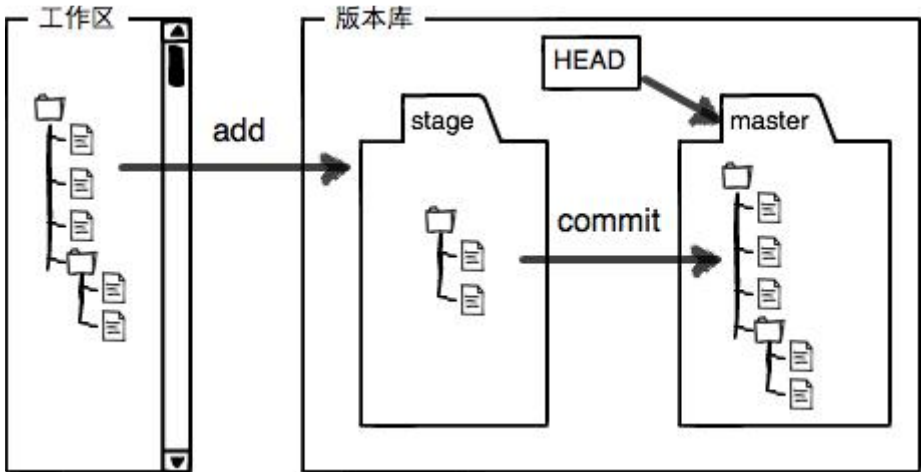
* 版本库 (Repository)

工作目录中的 `.git` 目录就是一个 Git 版本库。

⇒ 实际上 `.git` 虽然在工作目录下，但是不算是工作目录的一部分。

Git 的**版本库**中存了许多东西，例如：

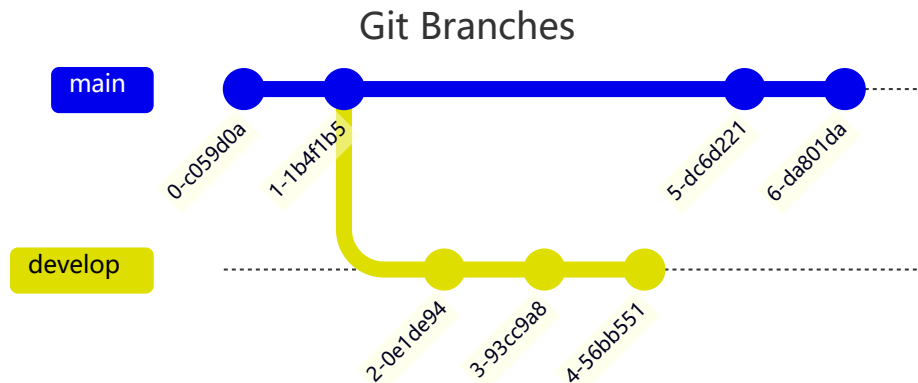
- `stage` (或叫 `index`) 的**暂存区**
- Git 自动创建的第一个分支(Branch) `master`
- 指向 `master` 的第一个指针 `HEAD`
- ...



在往 Git 版本库中添加文件的时候，实际上分两步：

1. 使用 `git add`，将文件添加到 **stage 暂存区**。
2. 使用 `git commit` 提交更改，将 **stage** 中的内容提交到**当前分支**(**master**)。

* 实际上**分支(Branch)**是有多种的，类似于以下的图样：



也正是 Git 强大的分支管理功能使得同一个项目能够以不同的版本共同推进。

当我们创建 Git 版本库的时候，Git 自动会为我们创建一个 **master** 分支，因此一般提交的时候，是默认在这个分支上进行版本迭代。

我们可以添加一个 **LICENSE** 文件(主要是编写开源协议的文件)，然后随便写一些文本进去。

```
This is a license file!  
This is only a license file!
```

然后将 **readme.txt** 文件进行修改：

```
Git Hello 11111111!  
VS Code Hello!  
Today is 2024/3/19
```

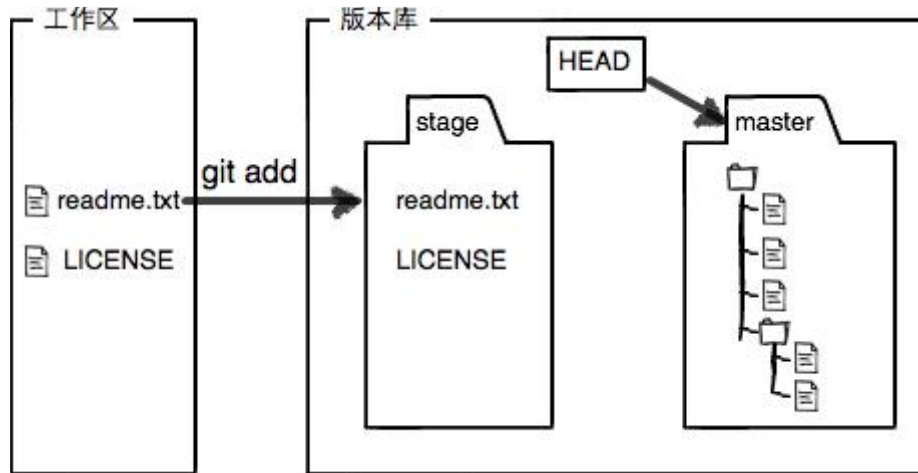
然后我们使用 `git status` 命令进行查：

```
$ git status  
On branch master  
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.txt  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    LICENSE  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

然后使用 `git add` 将 **readme.txt** 和 **LICENSE** 文件进行添加后，再次使用 `git status` 命令进行查看：


```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   LICENSE
    modified:   readme.txt
```

此时**版本库**的状态就是如下图所示：

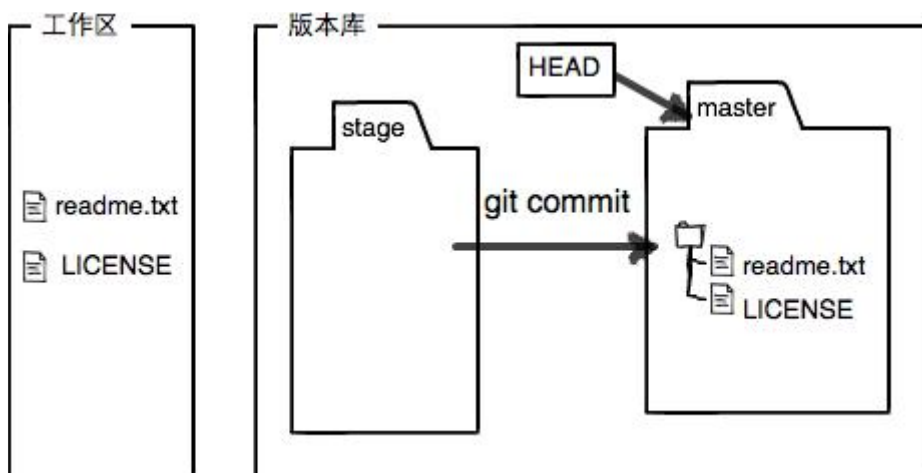


即，`git add` 命令就是将要提交的所有修改放到暂存区(Stage)。

然后我们执行 `git commit` 命令（别忘了注释）：

```
$ git commit -m "add a license and add date for readme"
[master 0aafe69] add a license and add date for readme
 2 files changed, 3 insertions(+)
 create mode 100644 LICENSE
```

此时版本库的状态如下图所示：



管理修改

Git 的优秀设计得益于它是**跟踪和管理修改**，而非**管理文件本身**。

我们可以做一个实验来验证 Git 的版本控制系统。

首先对 `readme.txt` 文件进行修改：

```
Git Hello 11111111!  
VS Code Hello!  
Today is 2024/3/19  
Challo~
```

然后，添加到**暂存区**：

```
$ git add readme.txt  
$ git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   readme.txt
```

然后，再次修改 `readme.txt` 文件：

```
Git Hello 11111111!  
VS Code Hello!  
Today is 2024/3/19  
Oops, No Challos ~
```

然后直接进行提交：

```
$ git commit -m "add Challos to readme"  
[master 3cb0ed7] add Challos to readme  
 1 file changed, 1 insertion(+)
```

再次查看状态：

```
$ git status  
On branch master  
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.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

显示**第二次修改并未被提交**。

我们回顾以下刚才的操作过程：

```
修改 readme.txt → git add → 再次修改 readme.txt → git  
commit
```

由于 Git 是管理修改的版本控制系统，因此使用 `git add` 命令后，第一次修改的内容放进了**暂存区**，而第二次修改的内容未通过 `git add` 放入**暂存区**。于是提交的时候只是提交**第一次的修改**。

我们可以使用 `git diff` 命令查看工作区和版本库中最新版本的区别：

```
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
index bd6329d..9c10368 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
   Git Hello 11111111!
   VS Code Hello!
   Today is 2024/3/19
-Challo~
\ No newline at end of file
+Oops, No Challos ~^M
```

可以发现，第二次修改并未被提交。

撤销修改

如果不小心修改了部分文件，但还未提交，我们先使用 `git status` 对工作区状态进行查看：

```
$ git status
On branch master
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.txt

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

这里提示我们可以使用 `git restore <file>` 去还原工作区内容：

```
$ git restore readme.txt
```

可以发现 `readme.txt` 文件被还原了。

同样，可以使用 `git checkout -- file` 达到相同的效果。

```
$ git checkout -- readme.txt
```

- 如果 `readme.txt` 自修改后未放到暂存区，那么就可以回退到和版本库一模一样的状态
- 如果 `readme.txt` 已经添加到了暂存区之后，又做了修改，则会回退到添加暂存区后的状态

总之，就是让这个文件回到最近一次 `git commit` 或 `git add` 时的状态。

如果不小心提交到暂存区了，那么可以使用 `git reset HEAD <file>` 将暂存区的修改撤销。

删除文件

在 Git 中，删除也是一个修改操作，先添加一个新文件 `test.txt` 到 Git 并提交。

```
$ git add test.txt
$ git commit -m "add test.txt"
[master 77c5d01] add test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

这时候，如果直接将这个文件给删除了：

```
$ rm test.txt
```

这个时候，Git 就知道你删除了文件，因此工作区和版本库就不一致了，使用 `git status` 命令可以得知哪些文件被删除了：

```
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    test.txt

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

现在有两个选择：

1. 删除文件 `git rm`，然后使用 `git commit`：

```
$ git rm test.txt
rm 'test.txt'
$ git commit -m "remove test.txt"
[master 7ca857b] remove test.txt
 1 file changed, 1 deletion(-)
delete mode 100644 test.txt
```

2. 删错了，那么使用 `git checkout/restore` 进行恢复：

```
$ git checkout -- test.txt
```

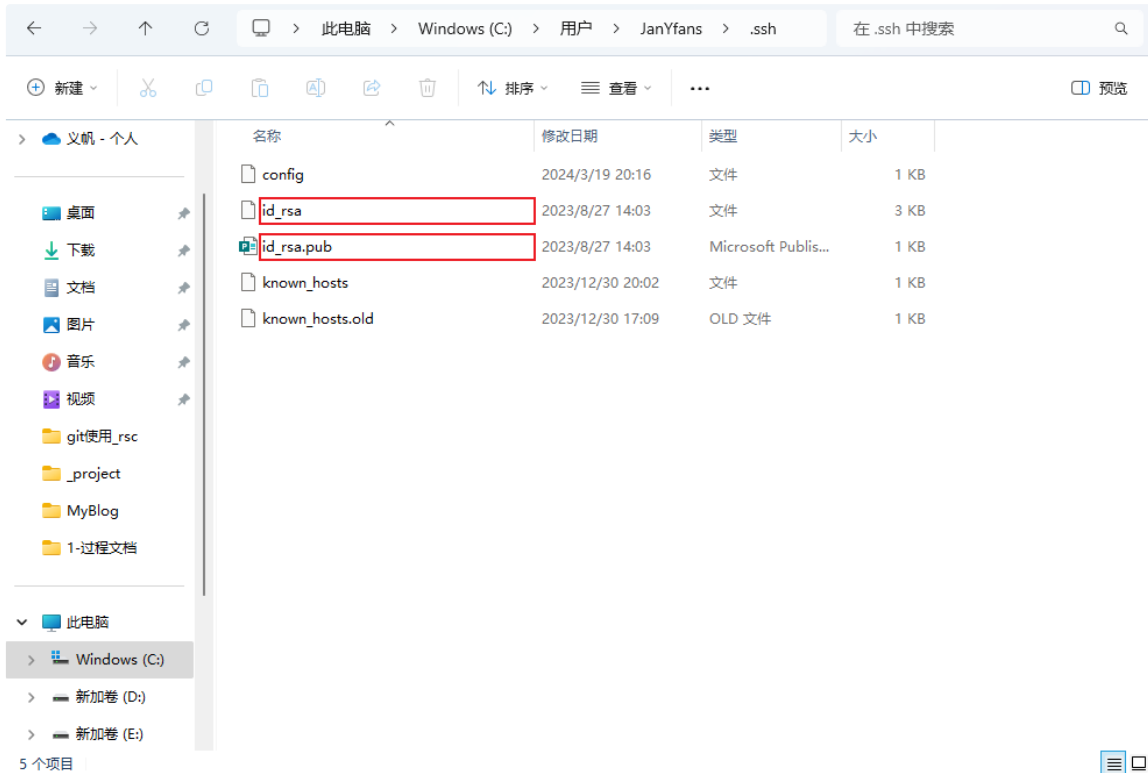
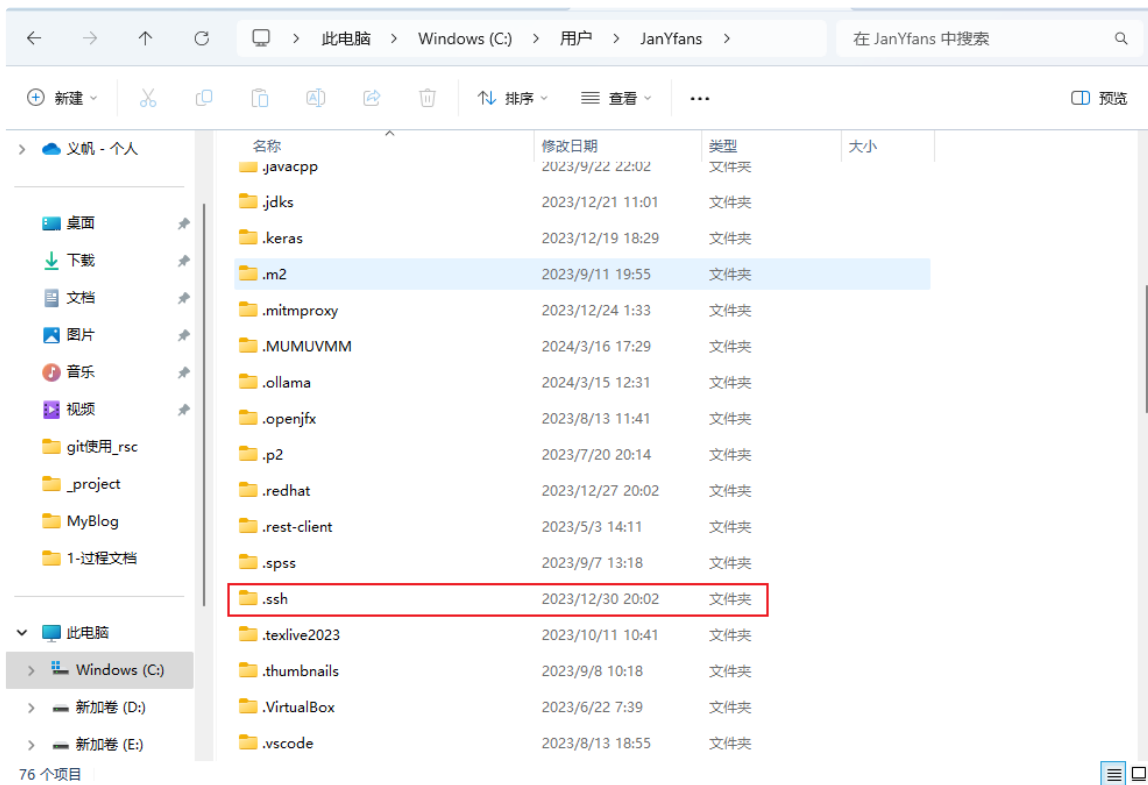
远程仓库

从这个部分开始为 Git 的核心部分之一，Git 作为一个分布式版本控制系统，使得同一个 Git 仓库可以分布到不同的机器上。

为了能够更加方便地实现 Git 的功能，GitHub 的出现让 Git 需要搭建服务器进行传输的压力骤然减小。这个网站主要是为 Git 提供仓库托管服务的。

由于本地 Git 仓库和 GitHub 仓库之间的传输是通过 SSH 加密的，因此需要一点设置：

1. 创建 SSH Key。在用户主目录下，创建（若无 `.ssh`）目录。**如果有这个目录**则查看是否有 `id_rsa` 和 `id_ras.pub` 两个文件，**如果有**则可以直接跳到**下一步**。如果没有，则打开 Git Bash，创建 SSH Key：



```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

需要填写自己的邮箱地址，然后一路回车即可。

如果完成的话，可以在用户主目录中找到 `.ssh` 目录，且里面含有 `id_rsa` 和 `id_rsa.pub` 两个文件。这两个就是 SSH Key 的密钥对，其中 `id_rsa` 为私钥，不能泄露，`id_rsa` 是公钥，可以放心地给任何人。

2. 登录 GitHub，打开"Settings"，"SSH and GPG keys"的页面：

Dashboard

Top Repositories

Find a repository...

janyfans/janyfans.github.io

janyfans/Comment_for_Blog

janyfans/test

janyfans/C-Code

Recent activity

When you take actions across GitHub, we'll provide links to that activity here.

Home

Updates to your homepage feed

We've combined the power of the Following feed with the For you feed so there's one place to discover content on GitHub. There's improved filtering so you can customize your feed exactly how you like it, and a shiny new visual design. ✨

Learn more

open-webui/open-webui released

2 days ago

v0.1.113

[0.1.113] - 2024-03-18

Added

Localization: You can now change the UI language in Settings > General. We support Ukrainian, German, Farsi (Persian), Traditional and Simplified Chinese and French ...

Read more

👤 1

❤️ 7

tbick labeled a pull request help wanted in open-webui/open-webui

2 weeks ago

feat(helm): adding lb class and labels for using it with Cilium CNI I... #1099

1-Merged

tbick merged 1 commit

...2 announcements made

Pull Request Checklist

Send feedback

Filter

Type [x] to search

The state of the Octob...

2023

What does it mean for a technol...

Discover how AI is changing the...

Learn mo...

Latest changes

15 hours ago

Enabment trends for se...

beta)

16 hours ago

Logging SAML SSO and S...

audit log events is gener...

2 days ago

Dependabot security upd...

registries even if target b...

5 days ago

Dependabot will now pau...

after 15 failures

View changelog —

Explore repositories

omagdy7 / ollama-logos

Logging plugin to integrate with o...

107

🔗 oscript

vallantynx / ollama-dock

Welcome to the Ollama Docker Co...

simplifies the deployment of Olla...

Janyfans

JanFans

Set status

Your profile

Add account

Your repositories

Your projects

Your Copilot

Your organizations

Your enterprises

Your stars

Your sponsors

Your gists

Upgrade

Try Enterprise

Feature preview

Settings

GitHub Docs

GitHub Support

GitHub Community

Sign out

<https://github.com/settings/ssh/new>

保存之后，可以输入 `ssh -T git@github.com` 查看是否成功。

添加远程库

如果想让本地的 Git 仓库和 GitHub 上进行同步，那么可以进行以下配置：


1. 登录 GitHub，然后找到"Top Repositories"，点击"New"。

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

 janyfans

Repository name *

Great repository names are short and memorable. Need inspiration? How about [expert-enigma](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None


A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository

然后直接点击 `Create repository` 即可。


2. 创建好后，我们注意这部分的内容，就是将本地仓库连接到 GitHub 上面。



Set up GitHub Copilot

Use GitHub's AI pair programmer to autocomplete suggestions as you code.

[Get started with GitHub Copilot](#)



Add collaborators to this repository

Search for people using their GitHub username or email address.

[Invite collaborators](#)

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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

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

...or push an existing repository from the command line

```
git remote add origin https://github.com/janyfans/testforgit.git
git branch -M main
git push -u origin main
```

...or import code from another repository

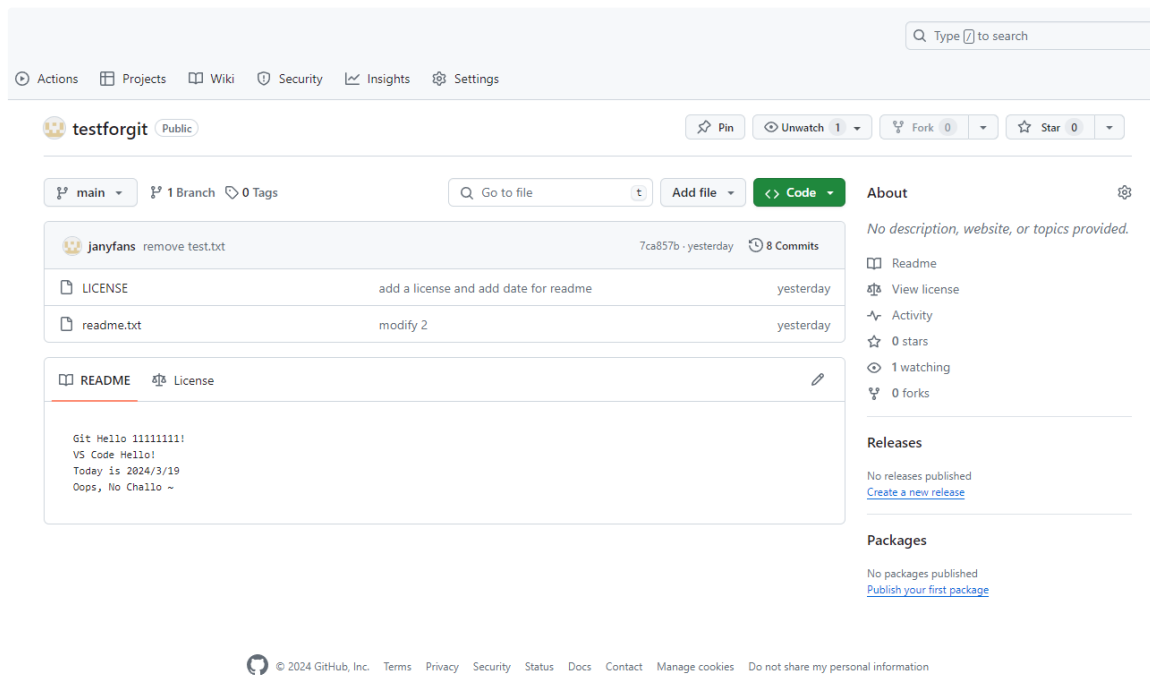
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

3. 在 git bash 上面输入:

```
$ git remote add origin https://github.com/janyfans/testforgit.git
$ git branch -M main
$ git push -u origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 20 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (23/23), 1.80 KiB | 924.00 KiB/s, done.
Total 23 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/janyfans/testforgit.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

刷新一下, 我们就可以发现我们的本地仓库部署到 GitHub 上面了。



- `git remote add origin` 是将本地仓库和 `origin` 进行链接。此处的 `origin` 指的是远程库的默认名称，改成其他的也可以。
- `git push -u origin master` 将本地库中的 `master` 分支推送到 GitHub 上。

4. 在之后本地提交之后，可以直接使用命令：

```
$ git push origin master
```

将本地的修改推送至 GitHub。

5. 如果想删除本地仓库和 GitHub 上仓库的链接，可以使用以下：

```
$ git remote rm origin
```

小结

要关联一个远程库，使用命令 `git remote add origin git@server-name:path/repo-name.git` ；

关联一个远程库时必须给远程库指定一个名字，`origin` 是默认习惯命名；

关联后，使用命令 `git push -u origin master` 第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令 `git push origin master` 推送最新修改；