

Git

作者：何光琴

学习路径：

<https://www.liaoxuefeng.com/wiki/896043488029600>

<https://www.runoob.com/git/git-tutorial.html>

http://backlogtool.com/git-guide/cn/intro/intro1_1.html

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

练习地址：https://learngitbranching.js.org/?locale=zh_CN

1.什么是Git

1.1 什么是Git

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

Git是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。Git是Linus Torvalds为了帮助管理Linux内核开发而开发的一个开放源码的版本控制软件。

1.2 Git的历史

与生活中的许多伟大事物一样，Git始于一些创造性的破坏和激烈的争论。

Linux内核是一个规模相当大的开源软件项目。在Linux内核维护的早期（1991-2002），对软件的更改以补丁和存档文件的形式传递。2002年，Linux内核项目开始使用名为BitKeeper的专有DVCS。

2005年，开发Linux内核的社区与开发BitKeeper的商业公司之间的关系破裂，该工具的免费状态被撤销。这促使Linux开发社区（尤其是Linux的创建者Linus Torvalds）根据他们在使用BitKeeper时学到的一些经验教训开发自己的工具。新系统的一些目标如下：

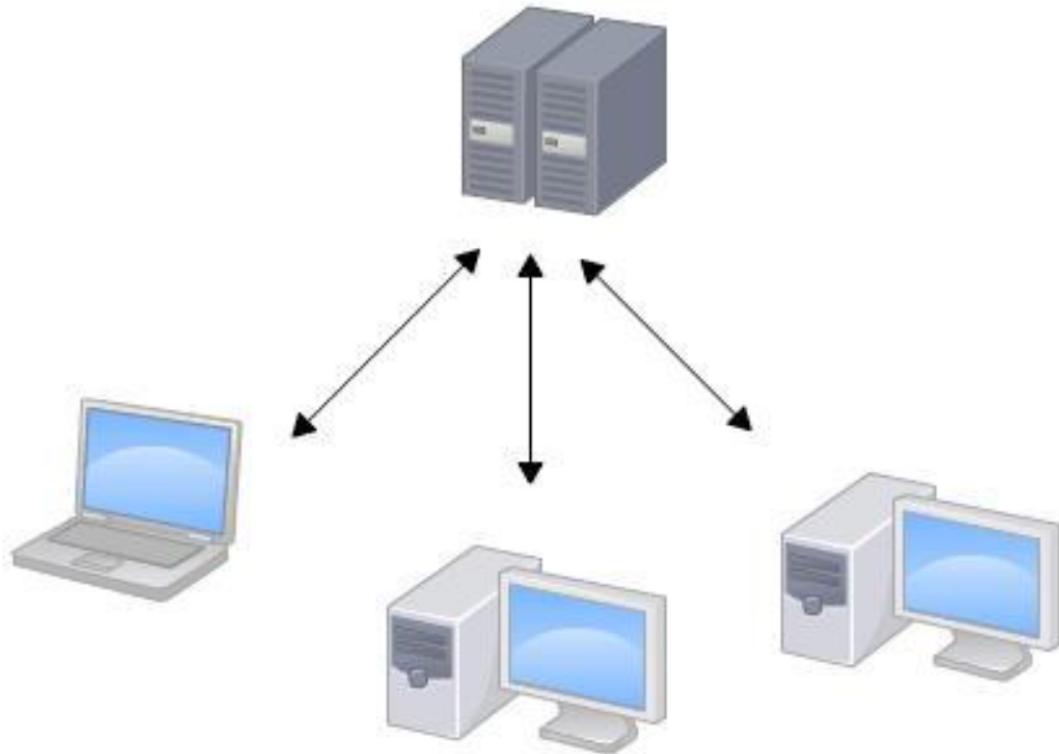
- 速度
- 简单的设计
- 对非线性开发的强大支持（数千个并行分支）
- 完全分布式
- 能够高效地处理像Linux内核这样的大型项目（速度和数据大小）

自2005年诞生以来，Git已经发展成熟，易于使用，同时保留了这些最初的品质。它的速度快得惊人，对于大型项目非常高效，并且它具有令人难以置信的非线性开发分支系统（请参阅[Git分支](#)）。

2. Git与SVN的区别

2.1 SVN

SVN是集中式版本控制系统，版本库是集中放在中央服务器的，而干活的时候，用的都是自己的电脑，所以首先要从中央服务器哪里得到最新的版本，然后干活，干完后，需要把自己做完的活推送到中央服务器。集中式版本控制系统是必须联网才能工作，如果在局域网还可以，带宽够大，速度够快，如果在互联网下，如果网速慢的话，就郁闷了。下图就是标准的集中式版本控制工具管理方式：

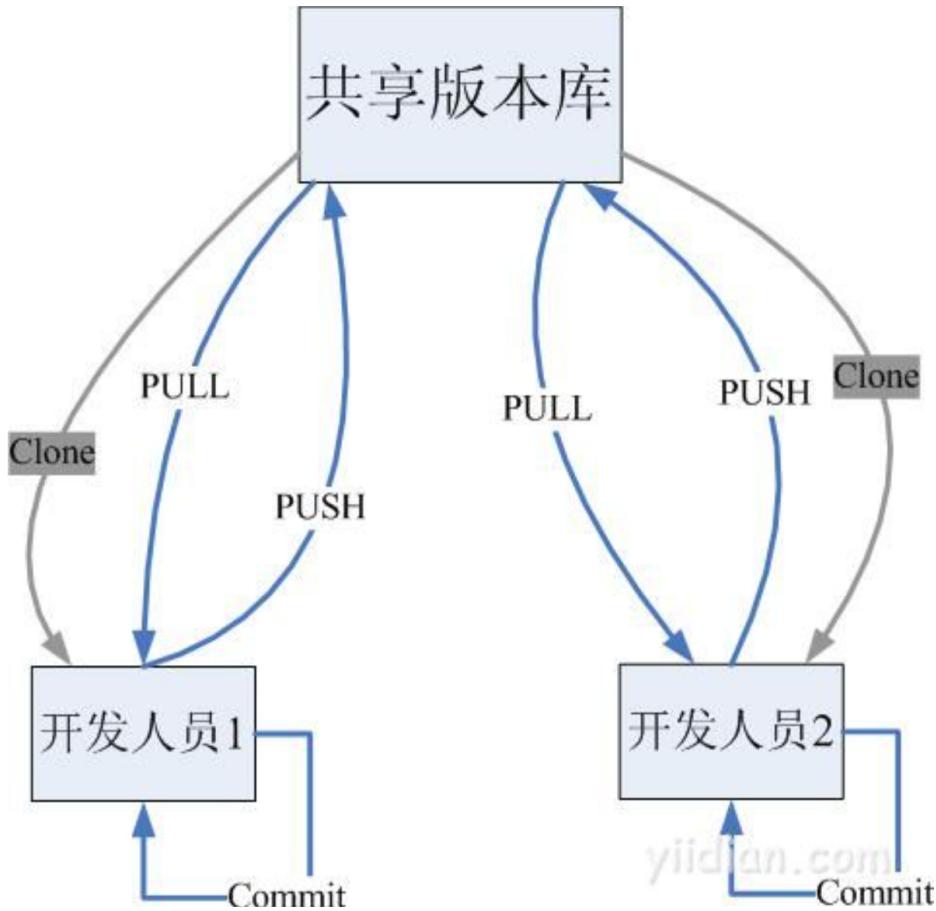


集中管理方式在一定程度上看到其他开发人员在干什么，而管理员也可以很轻松掌握每个人的开发权限。但是相较于其优点而言，集中式版本控制工具缺点很明显：

- 服务器单点故障
- 容错性差

2.2 Git

Git是分布式版本控制系统，那么它可以没有中央服务器的，每个人的电脑就是一个完整的版本库，这样，工作的时候就不需要联网了，因为版本都是在自己的电脑上。既然每个人的电脑都有一个完整的版本库，那多人如何协作呢？比如说自己在电脑上改了文件A，其他人也在电脑上改了文件A，这时，你们两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。下图就是分布式版本控制工具管理方式：



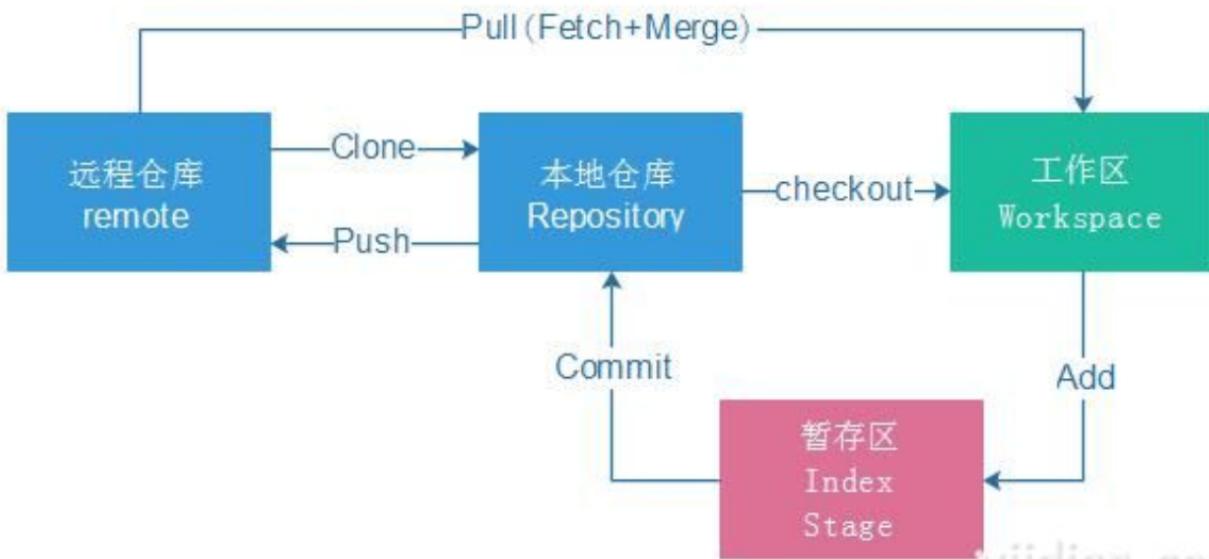
3. Git的工作流程

Git的工作流程如下：

1. 从远程仓库中克隆 Git 资源作为本地仓库。
2. 从本地仓库中checkout代码然后进行代码修改
3. 在提交前先将代码提交到暂存区。
4. 提交修改。提交到本地仓库。本地仓库中保存修改的各个历史版本。
5. 在修改完成后，需要和团队成员共享代码时，可以将代码push到远程仓库。

下图展示了Git的工作流程：

Git流程图

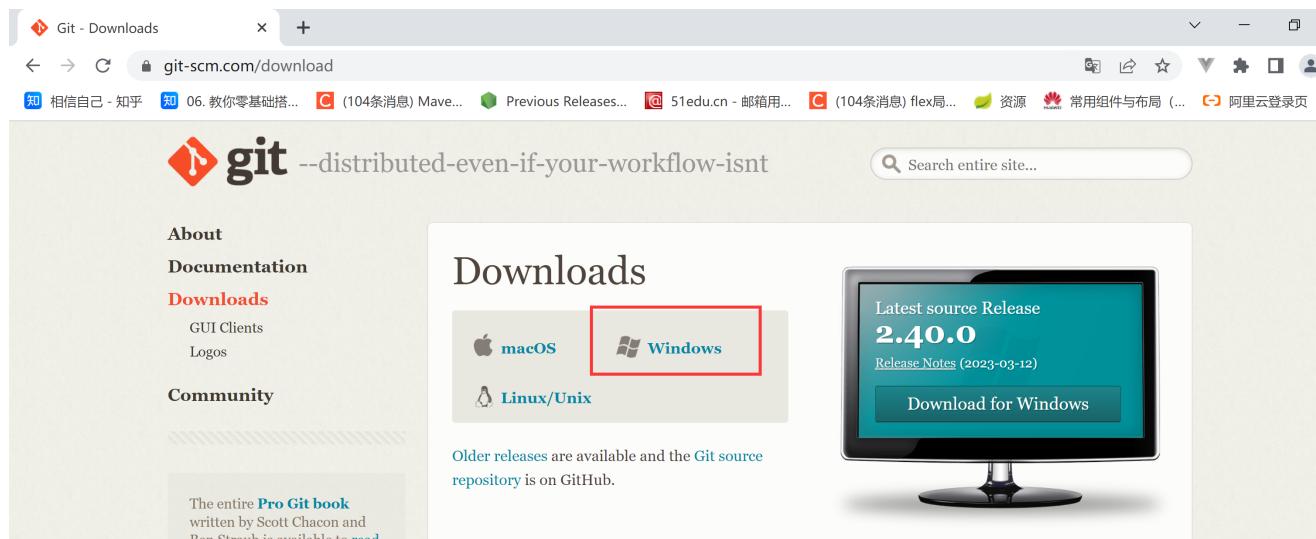


4. Git的安装

最早Git是在Linux上开发的，很长一段时间内，Git也只能在Linux和Unix系统上跑。不过，慢慢地有人把它移植到了Windows上。现在，Git可以在Linux、Unix、Mac和Windows这几大平台上正常运行了。由于开发机大多数情况都是windows，在这只讲解windows下的git的安装及使用。

4.1 Git下载

下载地址：<https://git-scm.com/download>



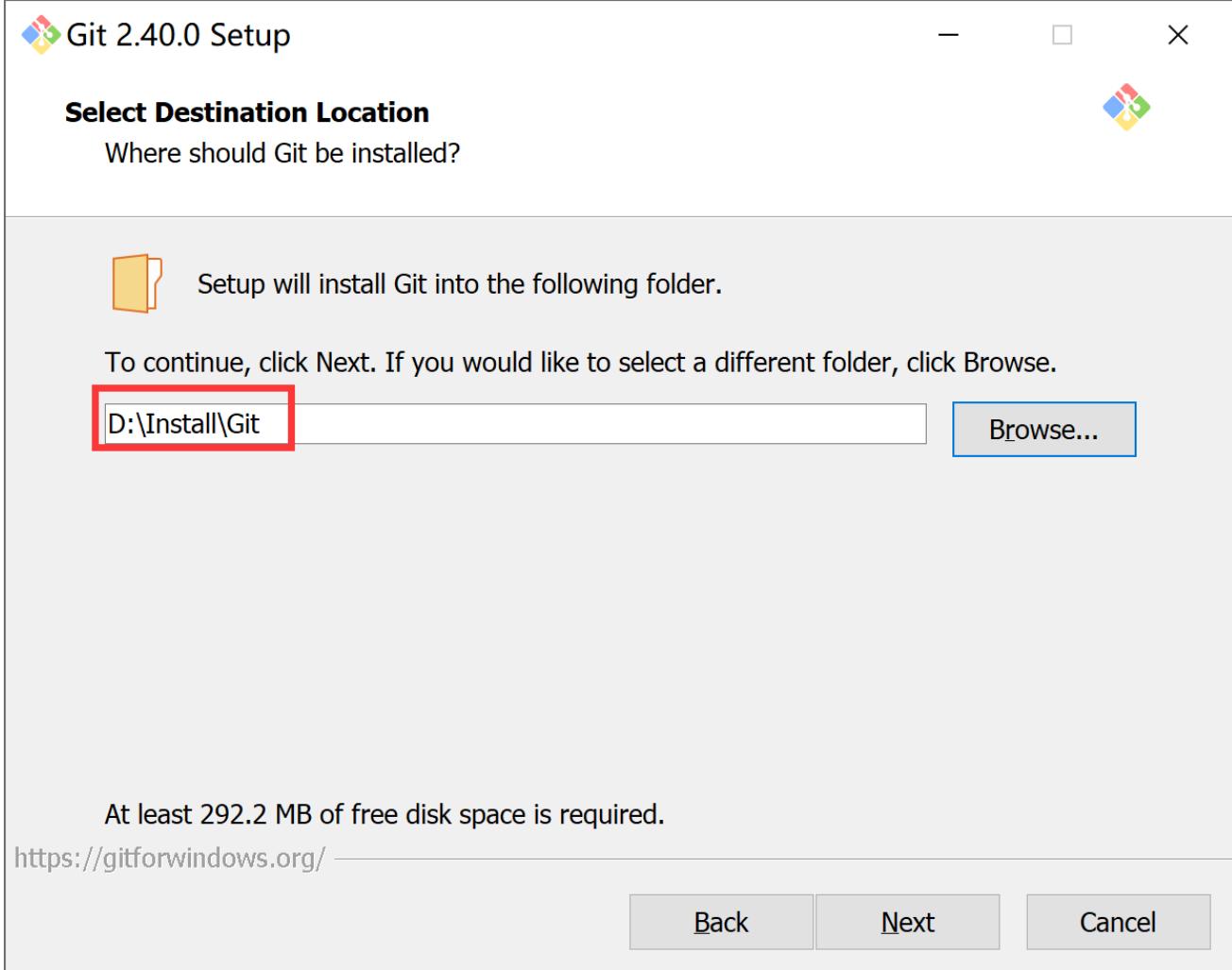
根据不同的操作系统选择对应的安装包。

4.2 安装Git

双击运行



选择安装目录



一路"Next"使用默认选项安装完成即可。

4.3 TortoiseGit下载及安装

下载地址: <https://tortoisegit.org/download/>

Git - 下载包 Download - TortoiseGit - Wind... 百度翻译-200种语言互译、沟通...

to tortoisegit.org/download/

相信自己 - 知乎 06. 教你零基础搭... C (104条消息) Maven... Previous Releases... 51edu.cn - 邮箱用... C (104条消息) flex局... 资源 常用组件与布局 (...) 阿里云登录页

 **TortoiseGit**
Windows Shell Interface to Git

About **Download** Get Support ▾ Contribute

TortoiseGit.org » Download

Download

The current stable version is: 2.14.0

For detailed info on what's new, read the [release notes](#).

FAQ: [System prerequisites and installation](#)

[Donate](#)

Please make sure that you choose the right installer for your PC, otherwise the setup will fail.

for 32-bit Windows	for 64-bit Windows	for ARM64 Windows
Download TortoiseGit 2.14.0 32-bit (18.4 MiB)	Download TortoiseGit 2.14.0 64-bit (21.6 MiB)	Download TortoiseGit 2.14.0 ARM64 (19.9 MiB)

Pre-Release Builds

Before reporting an issue, please check that your problem isn't fixed in our latest [preview release](#). Also see [What to do if a crash](#)

- 安装

双击运行

 Git-2.40.0-64-bit.exe	2023/3/30 21:22	应用程序
 TortoiseGit-2.14.0.0-64bit.msi	2023/3/30 21:57	Windows 安装文件
 TortoiseGit-LanguagePack-2.14.0.0-6...	2023/3/30 21:51	Windows 安装文件



TortoiseGit 2.14.0.0 (64 bit) Setup



Welcome to the TortoiseGit 2.14.0.0 (64 bit) Setup Wizard

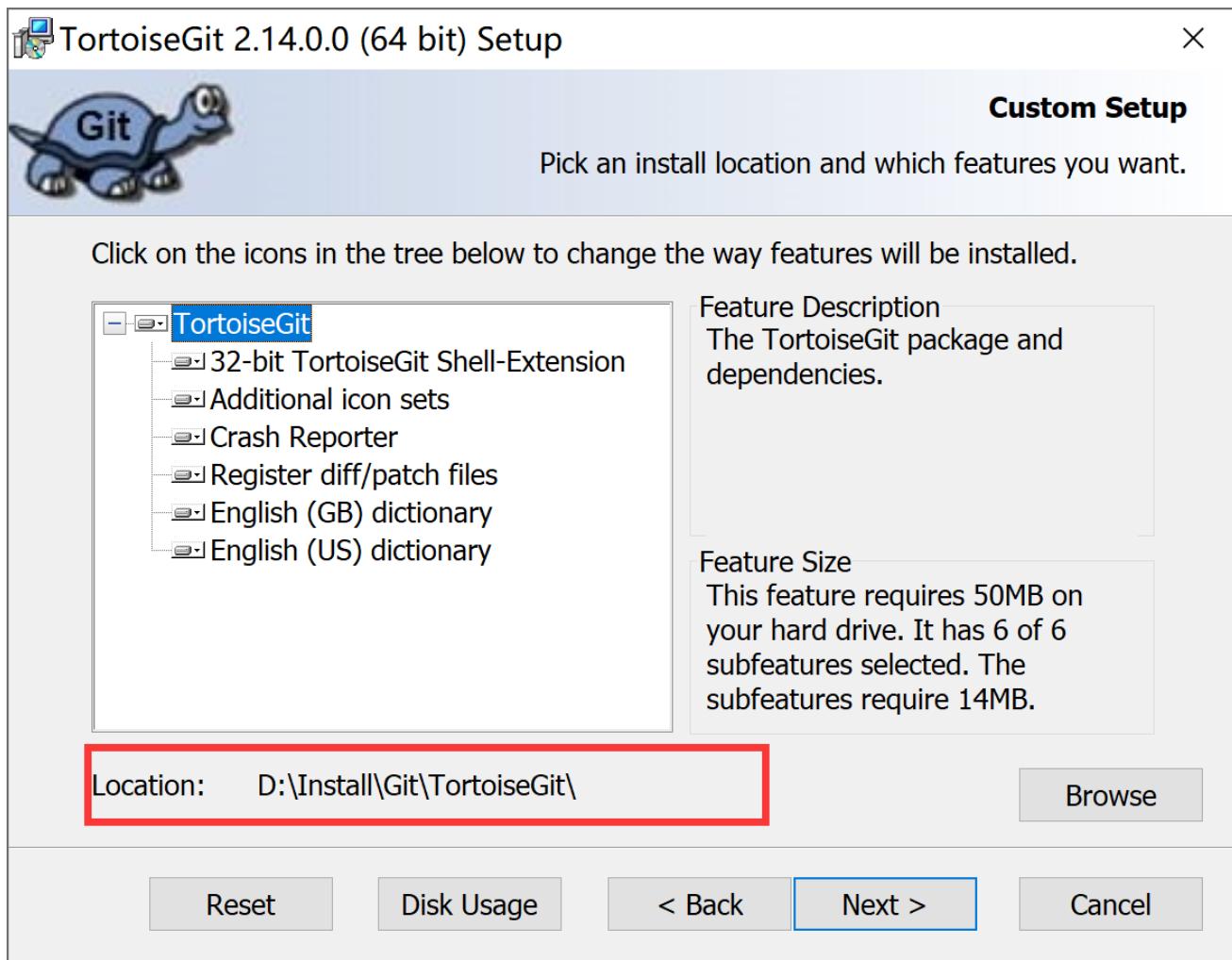
The Setup Wizard will install TortoiseGit 2.14.0.0 (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

< Back

Next >

Cancel

选择安装目录



一路“Next”使用默认选项安装完成即可。

- 安装中文语言包

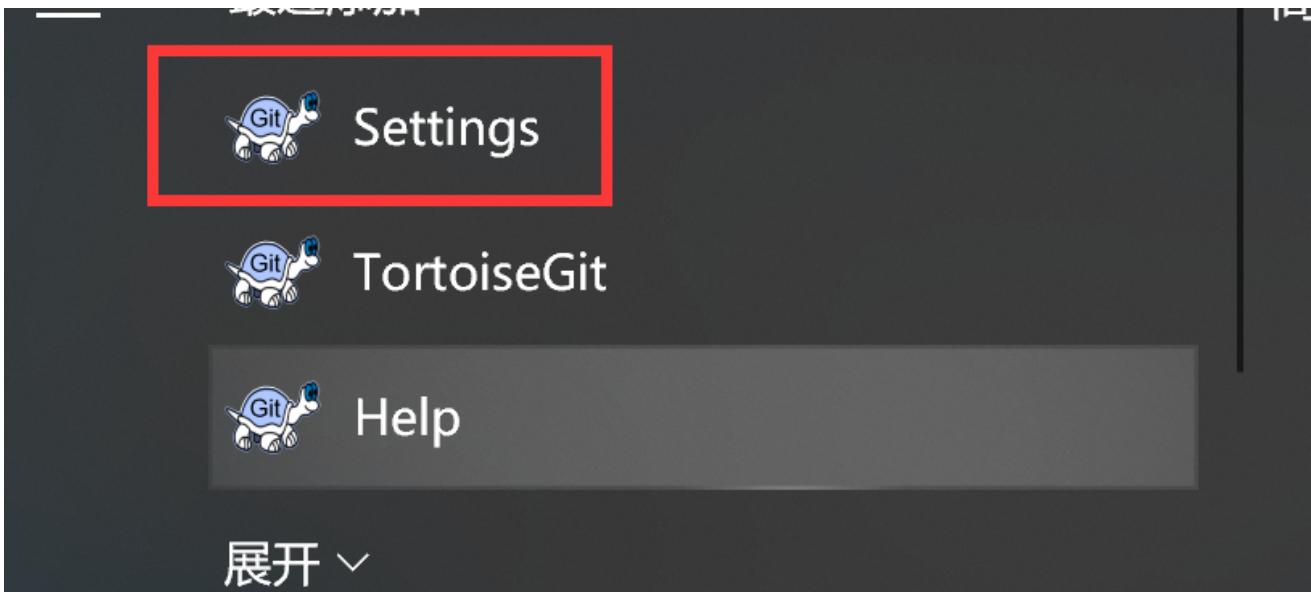
安装中文语言包并不是必选项。可以根据个人情况来选择安装。

双击打开。直接“下一步”完整完毕。

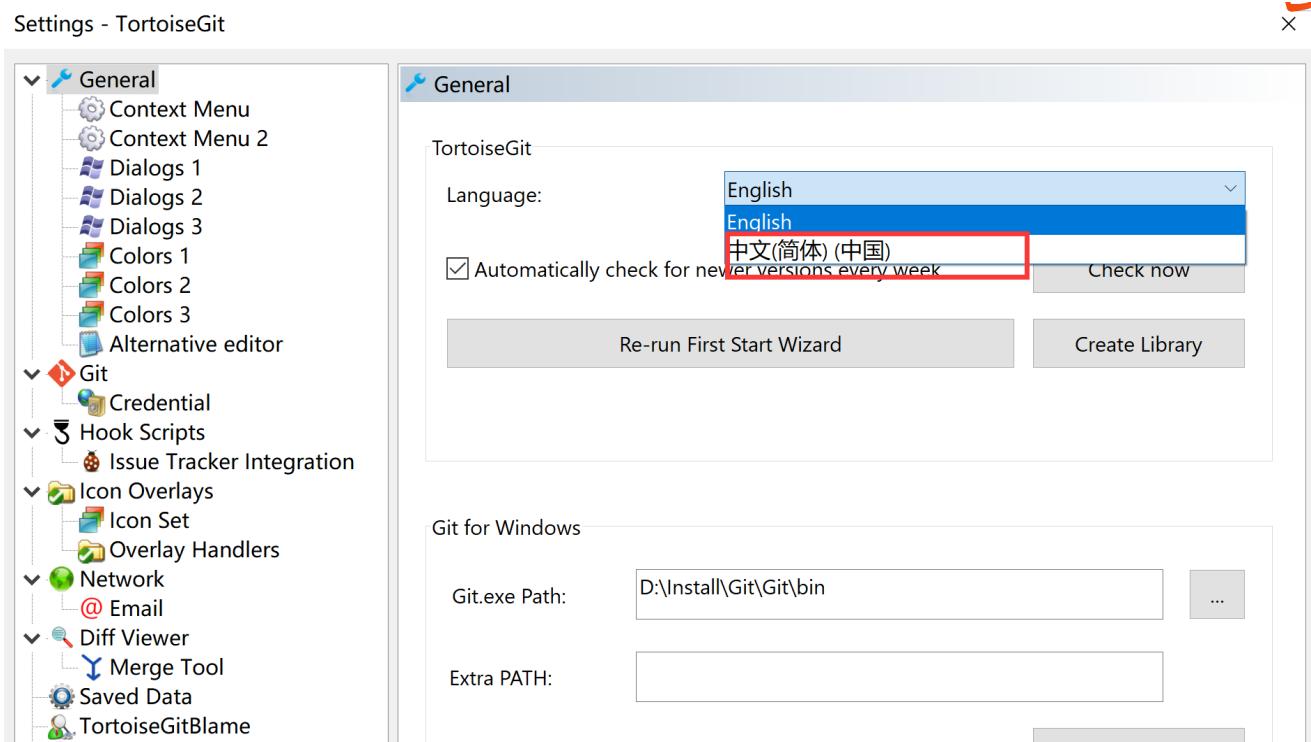
Git-2.40.0-64-bit.exe	2023/3/30 21:22	应用程序	5
TortoiseGit-2.14.0.0-64bit.msi	2023/3/30 21:57	Windows Installe...	2
TortoiseGit-LanguagePack-2.14.0.0-64bit-zh_CN.msi	2023/3/30 21:51	Windows Installe...	

4.4 配置

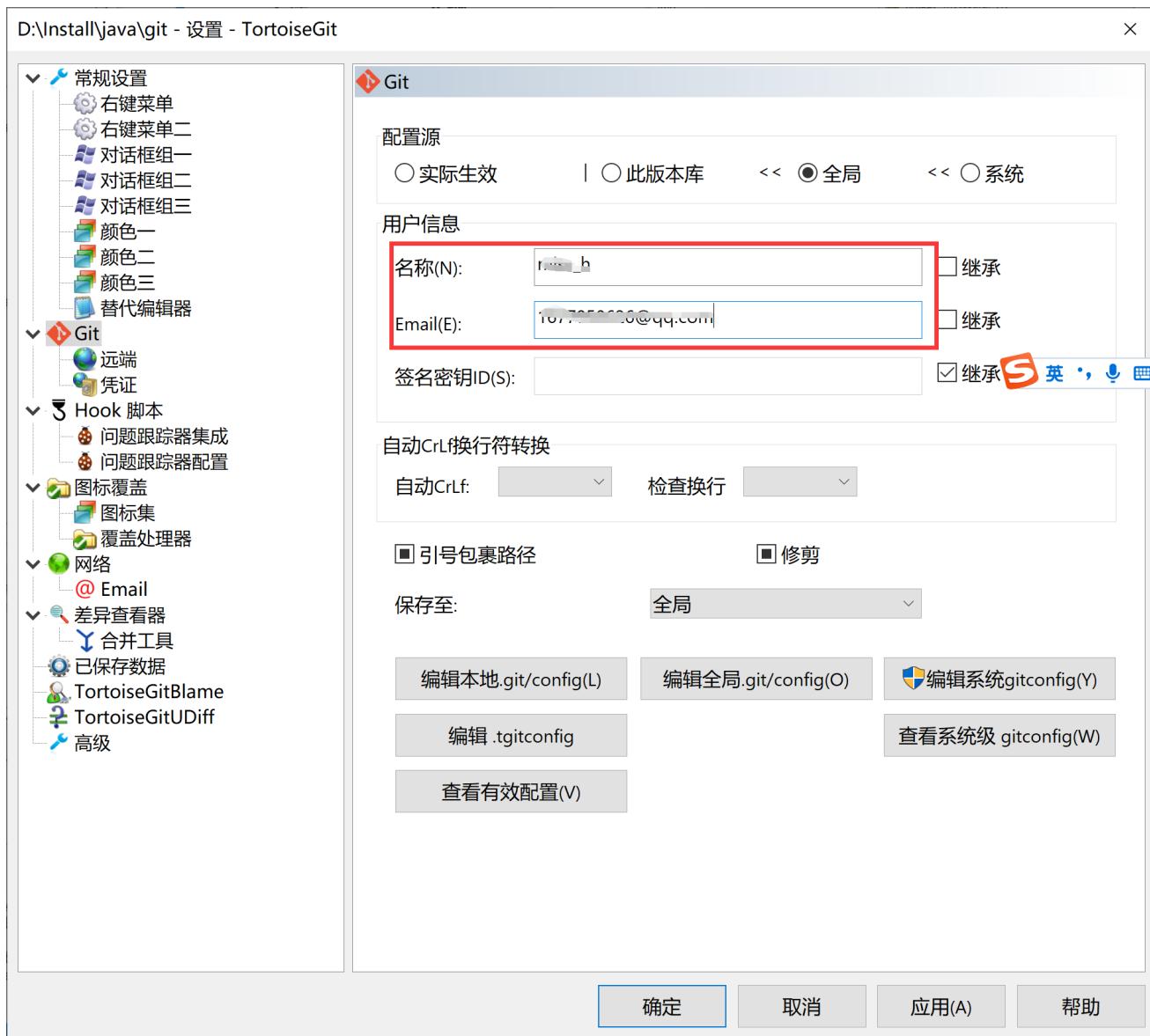
鼠标右键选择Settings



语言包安装完毕后可以在TortoiseGit的设置中调整语言。



配置用户名和邮箱



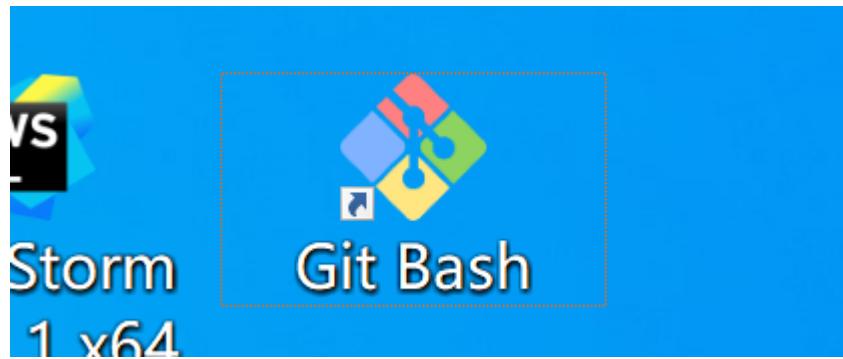
5. 创建版本库

什么是版本库呢？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”。由于git是分布式版本管理工具，所以git在不需要联网的情况下也具有完整的版本管理能力。

创建一个版本库非常简单，可以使用 `git bash` 也可以使用 `TortoiseGit`。首先，选择一个合适的地方，创建一个空目录（D:\Install\java\git）。

5.1 使用GitBash创建仓库

在当前目录中点击右键中选择Git Bash来启动



创建仓库执行命令：

```
$ git init
```

```
MINGW64:/d/Install/java/git

admin@DESKTOP-6D1Q115 MINGW64 ~
$ cd d://Install/java/git

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git
$ git init
Initialized empty Git repository in D:/Install/java/git/.git/
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```



版本库目录：“.git”目录就是版本库，将来文件都需要保存到版本库中。 **工作目录：**包含“.git”目录的目录，也就是.git目录的上一级目录就是工作目录。只有工作目录中的文件才能保存到版本库中。

6. Git的使用

6.1 工作区和暂存区的区别

Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。

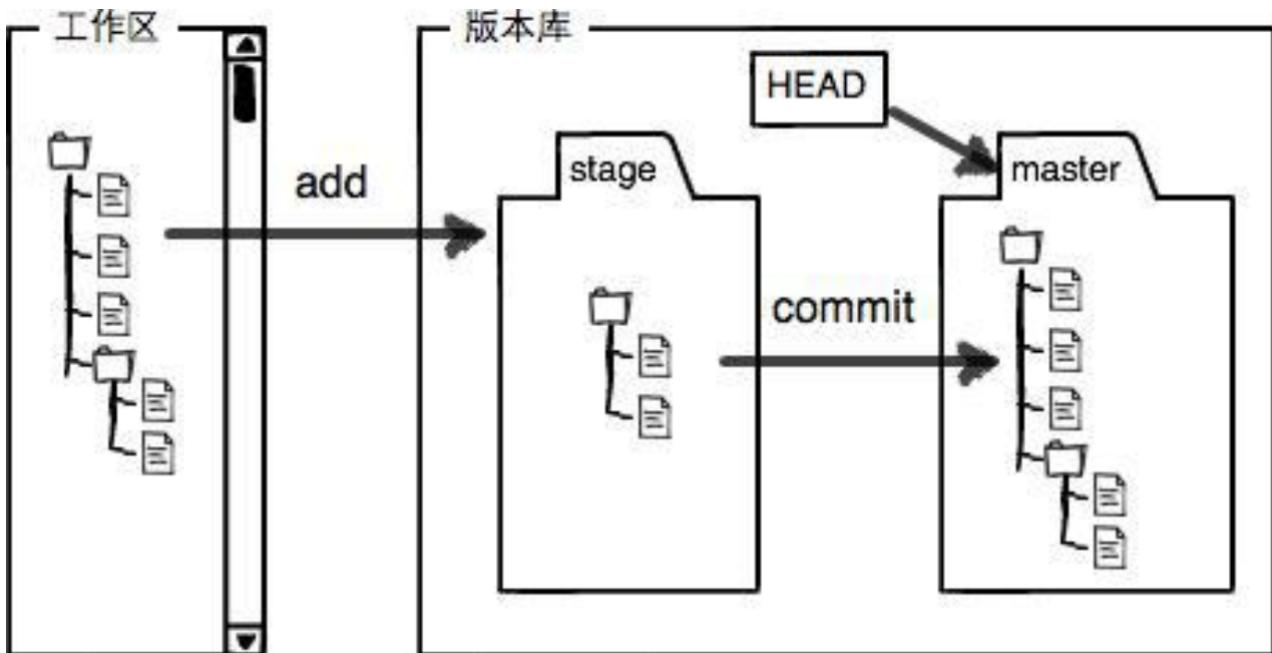
1. 什么是工作区

工作区就是你在电脑里能看到的目录，比如我的git文件夹就是一个工作区。

其实git目录是工作区，在这个目录中的“.git”隐藏文件夹才是版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

如下图所示：



分支和HEAD的概念我们后面文章再讲。前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

6.2 查看文件状态

在git的工作区添加文件，如下：

A screenshot of a file explorer window. The path is 'Data (D:) > Install > java > git'. The current view shows the contents of the '.git' folder. A red box highlights the contents of the '.git' folder, which include 'Demo.java' and 'mytest.txt'. The table below provides a detailed view of these files:

名称	修改日期	类型
.git	2023/3/31 15:14	文件
Demo.java	2022/12/8 10:47	JAV
mytest.txt	2023/3/31 14:48	文本

查看文件下所有文件的状态

```
git status
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git status
On branch master

No commits yet

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

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

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

一共有两种状态，红色表示当前状态是工作区，使用git add可以将其转换为缓冲区变为绿色。

6.3 将工作区的文件添加带缓存区

- 添加单个文件
- 添加所有文件

```
git add '文件名'
```

```
git add .
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git add . ← 添加所有文件
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git status ← 查看文件状态
On branch master
```

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
 new file: Demo.java
 new file: mytest.txt

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

6.4 将缓存区的文件提交到仓库区

```
git commit -m '版本一'
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git commit -m '版本一' ←
[master (root-commit) 6f74152] 版本一
 2 files changed, 11 insertions(+)
 create mode 100644 Demo.java
 create mode 100644 mytest.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

6.5 查看提交日志记录

```
git log
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git log
commit 6f741527bd6a9cda963774451e654bc184be17e8 (HEAD -> master)
Author: miss_h <1677850626@qq.com>
Date:   Fri Mar 31 15:20:51 2023 +0800

  版本一

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

红色框内是此次提交操作的ID，如果需要回滚时会使用，下面会介绍到。

6.6 修改

如果此时对文件进行修改，比如在txt文件中加入‘约饭’

然后此时在进行git status。

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ 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:   mytest.txt

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

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

监视到你对txt进行了修改，如果确定修改完毕，需要将其git add加载到缓冲区，然后就可以提交到仓库了。

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git add . ← 添加到缓存区

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git commit -m '版本二' ← 添加到仓库
[master 1eacaca] 版本二
 1 file changed, 3 insertions(+), 1 deletion(-)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

再次查看提交日志

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git log
commit 1eacaca37320dec582230964153c86f9e3c631bd (HEAD -> master)
Author: miss_h <1677850626@qq.com>
Date:   Fri Mar 31 15:28:38 2023 +0800
```

版本二

```
commit 6f741527bd6a9cda963774451e654bc184be17e8
Author: miss_h <1677850626@qq.com>
Date:   Fri Mar 31 15:20:51 2023 +0800
```

版本一

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

6.7 回滚

此时，发现问题，大事不好，需要回到上一个v1的版本，此时就需要学一个新的命令，回滚。

回滚，这里会用到上面提到的ID

```
git reset --hard ID
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git log
commit 1eacaca37320dec582230964153c86f9e3c631bd (HEAD -> master)
Author: miss_h <1677850626@qq.com>
Date:   Fri Mar 31 15:28:38 2023 +0800
```

版本二

```
commit 6f741527bd6a9cda963774451e654bc184be17e8
Author: miss_h <1677850626@qq.com>
Date:   Fri Mar 31 15:20:51 2023 +0800
```

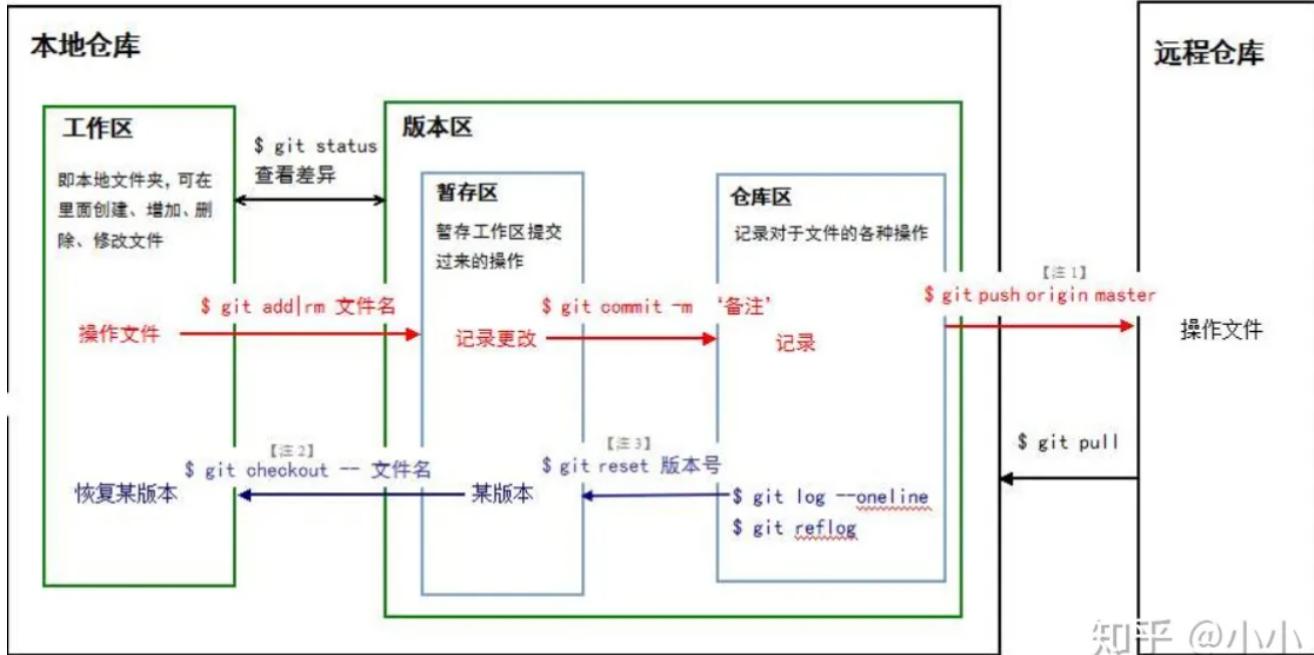
版本一

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git reset --hard 6f741527bd6a9cda963774451e654bc184be17e8
HEAD is now at 6f74152 版本一
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

head表示当前版本状态，红色框内表示两个版本的ID号，回滚到那个版本就是那个ID，现在我们回滚到版本一



6.7 删除

在Git中，删除也是一个修改操作

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用 `rm` 命令删了：

```
rm mytest.txt
```

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，`git status` 命令会立刻告诉你哪些文件被删除了：

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ rm mytest.txt ←

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git status ←
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:   mytest.txt

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

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

现在你有两个选择：

- 一是确实要从版本库中删除该文件，那就用命令 `git rm` 删掉，并且 `git commit`：

```
git rm mytest.txt

git commit -m "remove mytest.txt"
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git rm mytest.txt
rm 'mytest.txt'

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git commit -m "remove mytest.txt"
[master 301091d] remove mytest.txt
 1 file changed, 2 deletions(-)
 delete mode 100644 mytest.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```

现在，文件就从版本库中被删除了。

- 另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

```
git checkout -- Demo.java
```

`git checkout` 其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ rm Demo.java          ← 删除工作区文件，但是版本库中依然存在
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git status             ← 查看文件状态
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:   Demo.java

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

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git checkout -- Demo.java ← 从版本库恢复工作区已删除的文件
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```

小结：

命令 `git rm` 用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容。

Git常用命令：

`mkdir XX` (创建一个空目录 XX指目录名)

`pwd`：显示当前目录的路径。

`git init` 把当前的目录变成可以管理的git仓库，生成隐藏.git文件。

`git add XX` 把XX文件添加到暂存区去。

`git commit -m "XX"` 提交文件 -m 后面的是注释。

git status 查看仓库状态

git diff XX 查看XX文件修改了那些内容

git log 查看历史记录

git reset --hard HEAD^ 或者 git reset --hard HEAD~ 回退到上一个版本 (如果想回退到100个版本, 使用git reset --hard HEAD~100)

cat XX 查看XX文件内容

git reflog 查看历史记录的版本号id

git checkout -- XX 把XX文件在工作区的修改全部撤销。

git rm XX 删除XX文件

git remote add origin <https://github.com/tugenhua0707/testgit> 关联一个远程库

git push -u(第一次要用-u 以后不需要) origin master 把当前master分支推送到远程库

git clone <https://github.com/tugenhua0707/testgit> 从远程库中克隆

git checkout -b dev 创建dev分支 并切换到dev分支上

git branch 查看当前所有的分支

git checkout master 切换回master分支

git merge dev 在当前的分支上合并dev分支

git branch -d dev 删除dev分支

git branch name 创建分支

git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作

git stash list 查看所有被隐藏的文件列表

git stash apply 恢复被隐藏的文件, 但是内容不删除

git stash drop 删除文件

git stash pop 恢复文件的同时 也删除文件

git remote 查看远程库的信息

git remote -v 查看远程库的详细信息

git push origin master Git会把master分支推送到远程库对应的远程分支上

8. 远程仓库

到目前为止, 我们已经掌握了git的基本操作增删改查。

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。

你肯定会想，至少需要两台机器才能玩远程库不是？但是我只有一台电脑，怎么玩？

其实一台电脑上也是可以克隆多个版本库的，只要不在同一个目录下。不过，现实生活中是不会有人这么傻的在一台电脑上搞几个远程库玩，因为一台电脑上搞几个远程库完全没有意义，而且硬盘挂了会导致所有库都挂掉，所以我也不告诉你在一台电脑上怎么克隆多个仓库。

实际情况往往就是这样，找一台电脑充当服务器的角色，每天24小时开机，其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上，并且各自把各自的提交推送到服务器仓库里，也从服务器仓库中拉取别人的提交。

完全可以自己搭建一台运行Git的服务器，不过现阶段，为了学Git先搭个服务器绝对是小题大作。好在这个世界上有个叫[GitHub](#)的神奇的网站，从名字就可以看出，这个网站就是提供Git仓库托管服务的，所以，只要注册一个GitHub账号，就可以免费获得Git远程仓库。

8.1 配置远程仓库

1. 什么是ssh

SSH是英文Secure Shell的简写形式。通过使用SSH，你可以把所有传输的数据进行加密，这样“中间人”这种攻击方式就不可能实现了，而且也能够防止DNS欺骗和IP欺骗。

使用SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。SSH有很多功能，它既可以代替Telnet，又可以为FTP、Pop、甚至为PPP提供一个安全的“通道”。

如果一个用户从本地计算机，使用SSH协议登录另一台远程计算机，我们就可以认为，这种登录是安全的，即使被中途截获，密码也不会泄露。

最早的时候，互联网通信都是明文通信，一旦被截获，内容就暴露无疑。1995年，芬兰学者Tatu Ylonen设计了SSH协议，将登录信息全部加密，成为互联网安全的一个基本解决方案，迅速在全世界获得推广，目前已经成为Linux系统的标准配置。

2. 生成ssh秘钥

首先，自行注册GitHub账号。由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

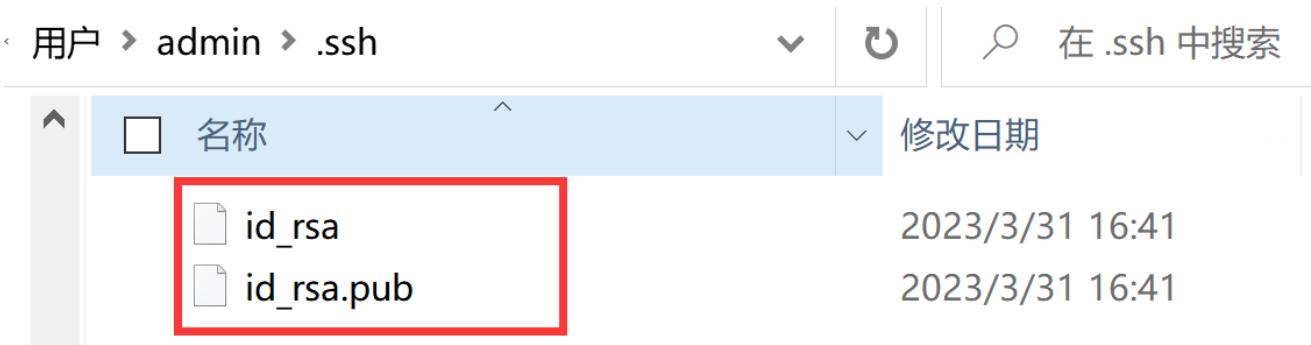
创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有 `id_rsa` 和 `id_rsa.pub` 这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

```
ssh-keygen -t rsa
```

```
MINGW64:/c/Users/admin
admin@DESKTOP-6D1Q115 MINGW64 ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/admin/.ssh/id_rsa):
Created directory '/c/Users/admin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/admin/.ssh/id_rsa
Your public key has been saved in /c/Users/admin/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:eQ2H3irCwSrIFz+t75gM9kBKbtIQjqToduhYEEiojJg admin@DESKTOP-6D1Q115
The key's randomart image is:
+---[RSA 3072]---+
| .
| O.
| E+     o .
| =O .   o =
| =O o   o S o o
| *=+ oo... .
| oX+^oo...
| *.=.=. .
| .. *oo
+---[SHA256]---+
admin@DESKTOP-6D1Q115 MINGW64 ~
$ |
```

然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里找到 `.ssh` 目录，里面有 `id_rsa` 和 `id_rsa.pub` 两个文件，这两个就是 SSH Key 的秘钥对，`id_rsa` 是私钥，不能泄露出去，`id_rsa.pub` 是公钥，可以放心地告诉任何人。



3. 在github配置ssh秘钥

密钥生成后需要在github上配置密钥本地才可以顺利访问

登陆GitHub，打开“settings”，“SSH and GPG Keys”页面：

A screenshot of a web browser showing the GitHub 'Public profile' settings page. The URL in the address bar is `github.com/settings/profile`. On the left, there's a sidebar with various account management options like 'Account', 'Appearance', and 'Notifications'. The main area shows fields for 'Name', 'Public email', 'Bio', 'Pronouns', and 'URL'. To the right is a 'Profile picture' section with a placeholder image of a pink 'H'. A red arrow points from the top right towards a context menu that is overlaid on the page. The menu is titled 'Signed in as H-Twenty' and includes links such as 'Your profile', 'Your repositories', 'Your projects', etc., followed by 'Settings' which is highlighted with a blue background.

然后，点“SSH and GPG Keys”，填上任意Title，在Key文本框里粘贴 `id_rsa.pub` 文件的内容

A screenshot of the GitHub 'SSH and GPG Keys' settings page. The URL in the address bar is `https://github.com/settings/keys`. The sidebar on the left has a red box around the 'SSH and GPG keys' option. The main content area shows sections for 'SSH keys' and 'GPG keys', each with a 'New [key type] key' button. A red arrow points from the bottom left towards the 'New SSH key' button in the 'SSH keys' section.

The screenshot shows the GitHub 'Add new SSH keys' interface. On the left, there's a sidebar with account settings like Public profile, Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions), SSH and GPG keys (selected), Organizations, and Moderation. The main area is titled 'SSH keys / Add new'. It has fields for 'Title' (set to 'myapplication'), 'Key type' (set to 'Authentication Key'), and a large text area for 'Key' containing an SSH RSA key. A red arrow points to the 'Add SSH key' button at the bottom left of this area.

点“Add SSH Key”，你就应该看到已经添加的Key：

The screenshot shows the GitHub 'SSH keys' page. The sidebar is identical to the previous one. The main area lists 'Authentication Keys' with a single entry: 'myapplication' (SHA256: eQ2H3irCwSrIFz+t75gM9kBKbtIQjqToduhYEE1oJg, Added on Mar 31, 2023, Never used — Read/write). A red box highlights this entry. Below it is a section for 'GPG keys' which is currently empty. A red arrow points to the 'New SSH key' button at the top right of the 'Authentication Keys' section.

为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是自己推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到（但只有你自己才能改）。所以，不要把敏感信息放进去。

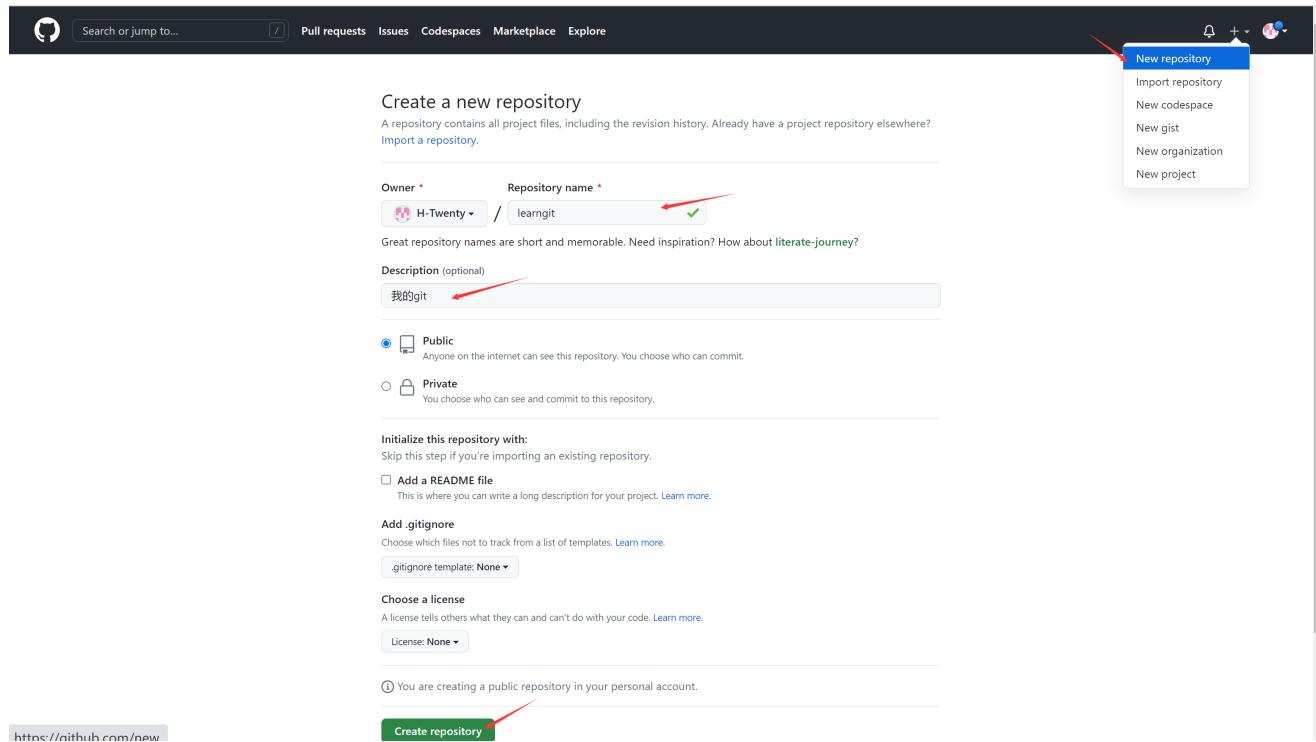
如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不到了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

确保你拥有一个GitHub账号后，我们就即将开始远程仓库的学习。

8.2 添加远程仓库

现在的情景是，你已经在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作，真是一举多得。

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：



在Repository name填入 `learngit`，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：

The screenshot shows the GitHub repository setup page for 'H-Twenty / learngit'. At the top, there's a search bar and navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below the header, the repository name 'H-Twenty / learngit' is shown with a 'Public' status. A red arrow points to the URL 'https://github.com/H-Twenty/learngit.git' in the 'Set up in Desktop' section.

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

Set up in Desktop or [HTTPS](https://github.com/H-Twenty/learngit.git) [SSH](https://github.com/H-Twenty/learngit.git) <https://github.com/H-Twenty/learngit.git>

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

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

```
git remote add origin https://github.com/H-Twenty/learngit.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](#)

ProTip! Use the URL for this page when adding GitHub as a remote.

目前，在GitHub上的这个 `learngit` 仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

现在，我们根据GitHub的提示，在本地的 `learngit` 仓库下运行命令：

```
$ git remote add origin git@github.com:H-Twenty/learngit.git
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)  
$ git remote add origin git@github.com:H-Twenty/learngit.git ↵  
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)  
$ |
```

请千万注意，把上面的 `H-Twenty` 替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是 `origin`，这是Git默认的叫法，也可以改成别的，但是 `origin` 这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

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

```

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git remote add origin git@github.com:H-Twenty/learngit.git

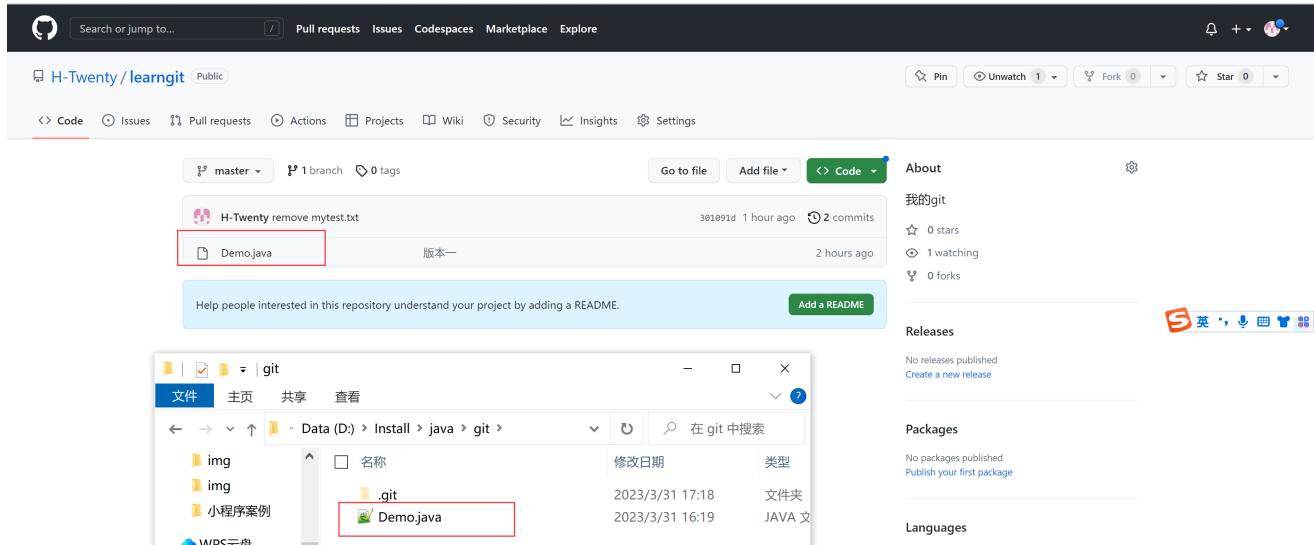
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git push -u origin master ←
The authenticity of host 'github.com (<no hostip for proxy command>)' can't be e
stablished.
ED25519 key fingerprint is SHA256:+DiY3vvV6TuJJhbpZisF/zLDA0zPMsvHdkr4UvCoqu.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 610 bytes | 610.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:H-Twenty/learngit.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```

把本地库的内容推送到远程，用 `git push` 命令，实际上是把当前分支 `master` 推送到远程。

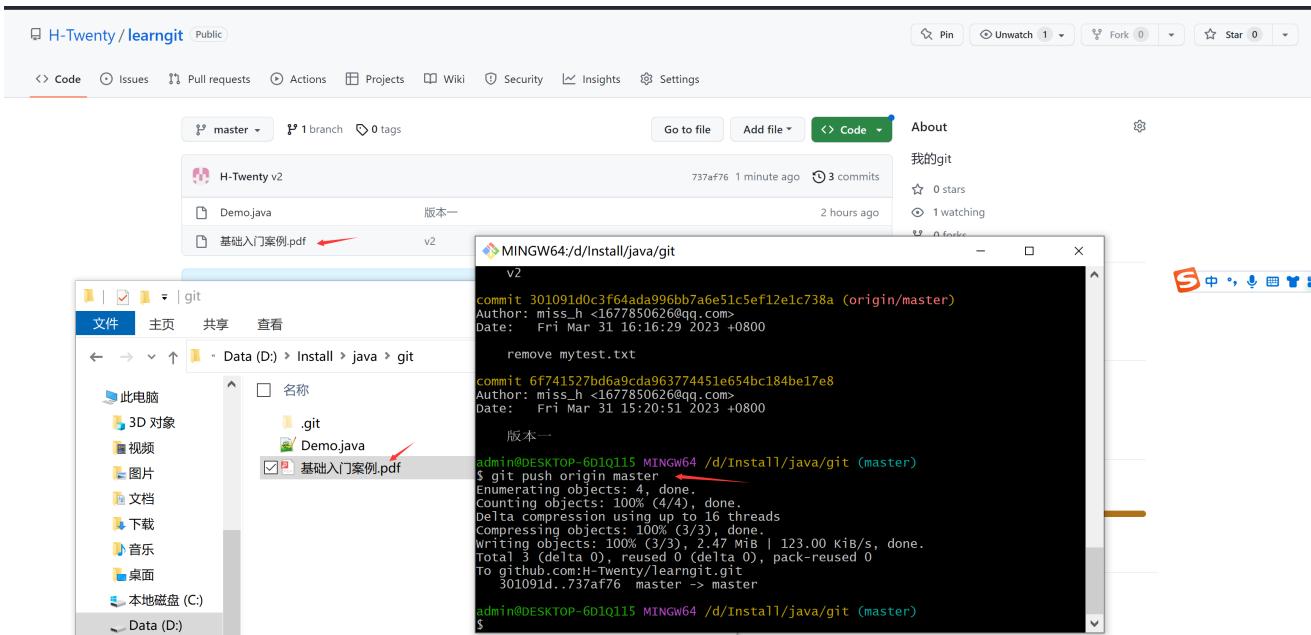
由于远端库是空的，我们第一次推送 `master` 分支时，加上了 `-u` 参数，Git不但会把本地的 `master` 分支内容推送到远端新的 `master` 分支，还会把本地的 `master` 分支和远端的 `master` 分支关联起来，在以后的推送或者拉取时就可以简化命令。

推送成功后，可以立刻在GitHub页面中看到远端库的内容已经和本地一模一样：



从现在起，只要本地作了提交，就可以通过命令：

```
$ git push origin master
```



把本地 `master` 分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！

注意：

SSH警告

当你第一次使用Git的 `clone` 或者 `push` 命令连接GitHub时，会得到一个警告：

```
The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.  
RSA key fingerprint is xx.xx.xx.xx.xx.  
Are you sure you want to continue connecting (yes/no)?
```

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入 `yes` 回车即可。

Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

```
Warning: Permanently added 'github.com' (RSA) to the list of known hosts.
```

这个警告只会出现一次，后面的操作就不会有任何警告了。

8.3 删 除 远 程 库

如果添加的时候地址写错了，或者就是想删除远 程 库，可以用 `git remote rm <name>` 命令。使用前，建议先用 `git remote -v` 查看远 程 库信息：

```
$ git remote -v
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git remote -v ←
origin  git@github.com:H-Twenty/learngit.git (fetch)
origin  git@github.com:H-Twenty/learngit.git (push)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```

然后，根据名字删除，比如删除 `origin`：

```
$ git remote rm origin
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git remote rm origin ←

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$
```

此处的“删除”其实是解除了本地和远程的绑定关系，并不是物理上删除了远程库。远程库本身并没有任何改动。要真正删除远程库，需要登录到GitHub，在后台页面找到删除按钮再删除。

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；

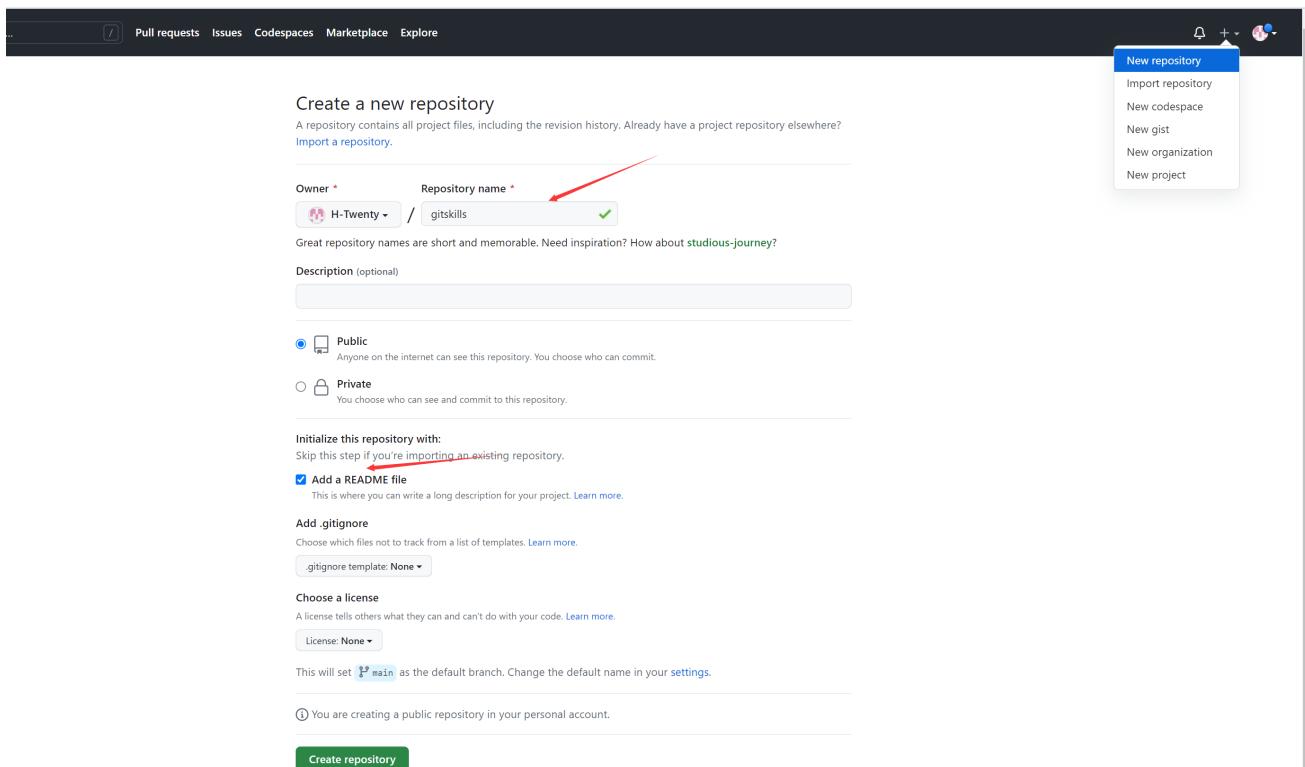
分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

8.4 从远程库克隆

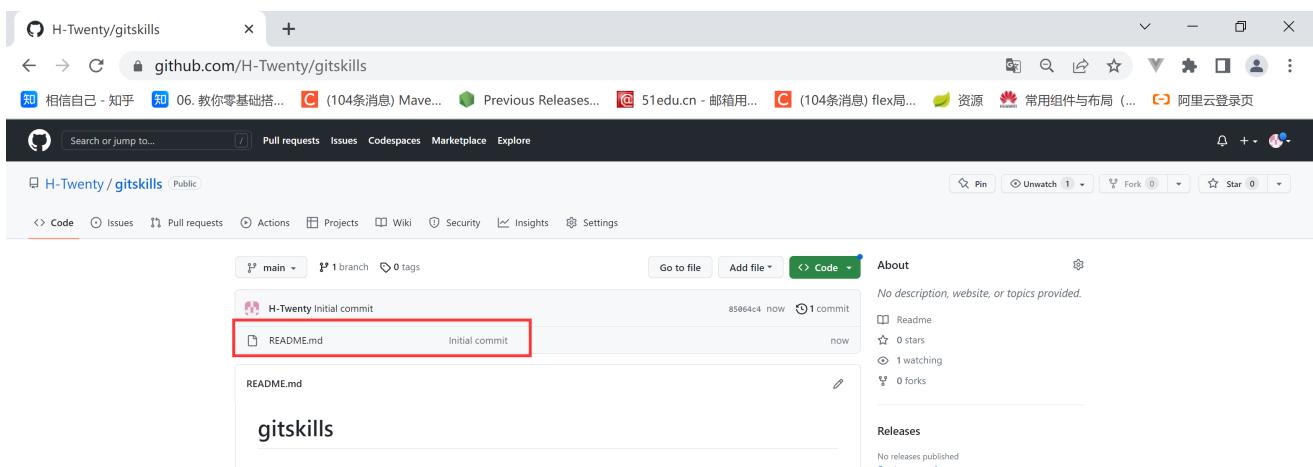
上次我们讲了先有本地库，后有远程库的时候，如何关联远程库。

现在，假设我们从零开发，那么最好的方式是先创建远程库，然后，从远程库克隆。

首先，登陆GitHub，创建一个新的仓库，名字叫 `gitskills`：



我们勾选 `Initialize this repository with a README`，这样GitHub会自动为我们创建一个 `README.md` 文件。创建完毕后，可以看到 `README.md` 文件：



现在，远端库已经准备好了，下一步是用命令 `git clone` 克隆一个本地库：

```
$ git clone git@github.com:H-Twenty/gitskills.git
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ git clone git@github.com:H-Twenty/gitskills.git
Cloning into 'gitskills'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)
$ |
```

注意把Git库的地址换成你自己的，然后进入 `gitskills` 目录看看，已经有 `README.md` 文件了：

```
$ cd gitskills  
$ ls
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git (master)  
$ cd gitskills ←  
  
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)  
$ ls  
README.md  
  
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)  
$
```

如果有多人协作开发，那么每个人各自从远程克隆一份就可以了。

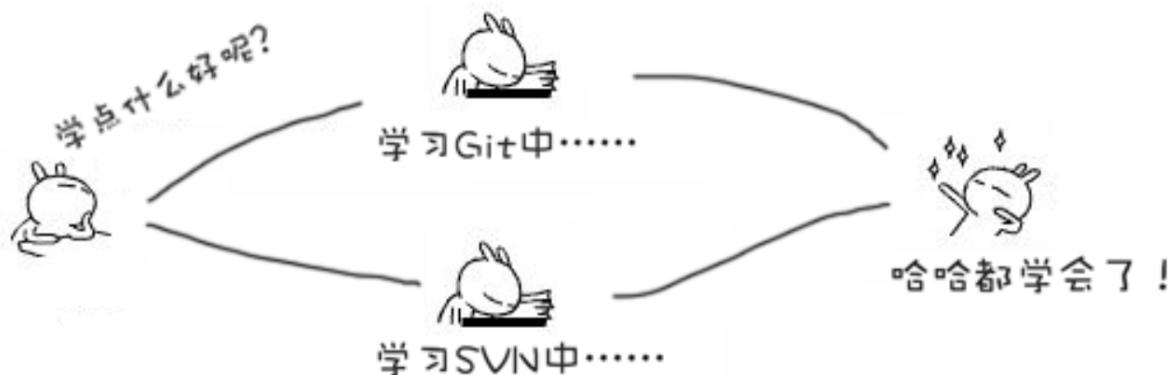
你也许还注意到，GitHub给出的地址不止一个，还可以用 <https://github.com/H-Twenty/gitskills.git> 这样的地址。实际上，Git支持多种协议，默认的 `git://` 使用ssh，但也可以使用 `https` 等其他协议。

使用 `https` 除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用 `ssh` 协议而只能用 `https`。

9. 分支管理

分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。

如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！



分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

9.1 分支相关操作

Git鼓励大量使用分支：

查看分支： `git branch`

创建分支： `git branch <name>`

切换分支： `git checkout <name>` 或者 `git switch <name>`

创建+切换分支： `git checkout -b <name>` 或者 `git switch -c <name>`

合并某分支到当前分支： `git merge <name>`

删除分支： `git branch -d <name>`

9.2 查看分支

```
git branch -v
```

目前我们有2个分支

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git branch -v
  dev  9ad7a87 add merge
* main 613e82e conflict fixed
```

9.3 创建分支

```
git branch 分支名
```

创建分支，并再次查看，发现对了一个分支

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git branch hot-fix 创建分支

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git branch -v ← 查看分支
  dev      9ad7a87 add merge
  hot-fix 613e82e conflict fixed
* main     613e82e conflict fixed

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$
```

9.4 切换分支

1. 切换分支

我们从上图最后一行末尾的蓝字还是“main”，说明还是在main分支上面的

```
git checkout 分支名

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git checkout hot-fix ← 切换分支
Switched to branch 'hot-fix'

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$
```

我们发现git后面的显示确实是从“main”变成了“hot-fix”

我们再查看一下分支，也确实改变了

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git branch -v
  dev      9ad7a87 add merge
* hot-fix 613e82e conflict fixed
  main     613e82e conflict fixed

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$
```

2. 修改代码并提交

```
vim readme.txt
```

我们可以查一下本地库状态

```
git status

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git status
On branch hot-fix
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")

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ |
```

添加到本地库

```
$ git add README.txt
```

再次查看

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git status
On branch hot-fix
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   readme.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
```

提交到本地库

```
$ git commit -m "hot-fix first commit" readme.txt
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git commit -m "hot-fix first commit" readme.txt
warning: in the working copy of 'readme.txt', LF will be replaced by CRLF
at time Git touches it
[hot-fix b983707] hot-fix first commit
 1 file changed, 3 insertions(+), 1 deletion(-)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
```

再次查看文件

```
$ cat readme.txt
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ cat readme.txt
Creating a new branch is quick and simple.
你好呀
好好好!!!
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
```

查看提交版本

```
$ git reflog
```

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git reflog
b983707 (HEAD -> hot-fix) HEAD@{0}: commit: hot-fix first commit
aa33c78 (main) HEAD@{1}: checkout: moving from main to hot-fix
aa33c78 (main) HEAD@{2}: merge hot-fix: Fast-forward
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{3}: checkout: moving from hot-fix to main
aa33c78 (main) HEAD@{4}: commit: hot-fix first commit
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{5}: checkout: moving from main to hot-fix
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{6}: checkout: moving from dev to main
9ad7a87 (dev) HEAD@{7}: checkout: moving from main to dev
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{8}: checkout: moving from dev to main
9ad7a87 (dev) HEAD@{9}: commit (initial): add merge
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{10}: checkout: moving from main to dev
613e82e (origin/main, origin/dev, origin/HEAD) HEAD@{11}: clone: from github.com:H-Twenty/gitskills.git

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$
```

9.5 合并分支

先将分支切换为main分支然后再合并

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git checkout main ←
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ |
```

然后我们查看一下代码，依然是原来的模样，因为当前指针又回来了，我们也没有合并

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ cat readme.txt
Creating a new branch is quick and simple.
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$
```

合并命令：

```
git merge 分支名
```

把指定的分支合并到当前分支上

1. 正常合并

下面这段代码的含义就是将“hot-fix”分支合并到“main”分支上

```
$ git merge hot-fix
```

```
Creating a new branch is quicker and simpler
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git merge hot-fix
Updating aa33c78..b983707
Fast-forward
 readme.txt | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
```

合并成功，一个文件被修改，两个删除，两行增加

[查看代码](#)

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ cat readme.txt
Creating a new branch is quick and simple.
你好呀
好好好!!!
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ |
```

2. 冲突合并（企业常见）

产生原因：合并分支时，两个分支在同一个文件的同一个位置有两套完全不同的修改，Git无法替我们决定使用哪一个，必需人为决定新代码内容

2.1 演示冲突情况（错误情况）

先修改以下main分支

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ vim readme.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ |
```

添加暂存区、提交本地库

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git add readme.txt ←

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git commit -m "mater test" readme.txt ←
[main 3886518] mater test
 1 file changed, 3 insertions(+)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ |
```

再修改以下hot-fix分支

切换hot-fix

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git checkout hot-fix
Switched to branch 'hot-fix'

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$
```

修改

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ vim readme.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ |
```

添加暂存区、提交本地库

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git add readme.txt ←
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git commit -m "hot-fix text" readme.txt ←
[hot-fix 918302c] hot-fix text
 1 file changed, 4 insertions(+)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
```

切换main开始操作

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (hot-fix)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ |
```

合并

合并时，提示冲突

翻译：

自动合并readme.txt

冲突（内容）：合并readme.txt中的冲突

自动合并失败；解决冲突，然后提交结果。

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ git merge hot-fix
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$
```

查看本地库状态

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  README.txt

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

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$
```

就告诉我们了，README.txt是没有合并成功的，两个分支都对他做了修改了

此时就不能自动合并了，就需要我们手动合并代码

2.2 手动修改

此时我们手动打开文件看一下，冲突的地方已经给我们标出来了

通过<<<< =====>>>> 这种特殊符号标记出来冲突

<<<HEAD 表示当前分支

在HEAD与=====之间是当前分支的代码

在=====与>>>>之间是我们要合并的代码

明显这两部分都做了修改，所以Git蒙蔽了

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ cat README.txt
Creating a new branch is quick and simple.
你好呀
好好好!!!
<<<<< HEAD
hello
hellohello
hellohelohello main
=====
haohaohaoaho
haohaohao
haohaohao
haohaohaoaho fix-hot text
>>>>> hot-fix

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$
```

如何手动合并？

想要把保存的留下来就可以了，手动删，把特殊符号也删掉，记得保存

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ vim readme.txt

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ |
```

此时我们应该把我们修改之后的文件放到暂存区，提交本地库（这个时候提交不能带文件名）

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ git add readme.txt ←

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main|MERGING)
$ git commit -m "merge test" ←
[main a15550a] merge test

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
```

完美查看文件

```
admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$ cat readme.txt
Creating a new branch is quick and simple.
你好呀
好好好!!!
hello
hellohello
hellohellohello man

admin@DESKTOP-6D1Q115 MINGW64 /d/Install/java/git/gitskills (main)
$
```