

1.Git基础

1.1 Git介绍

Git是目前世界上最先进的分布式 版本控制系统

版本控制系统：

设计师在设计的时候做了很多版本：



经过了数天去问设计师每个版本都改了啥，设计师此时可能就说不上来了。这个时候如果能有一个软件能记录每次的文件改动，并且还能协调多用户编辑，那岂不是美滋滋？这个软件应用起来应该像这个样子：

版本	文档名	操作用户	日志	修改时间
1	shejigao.txt	zhangsan	修改标题	2019-12-19 21:05:21
2	shejigao.txt	lisi	删除备注信息	2019-12-19 21:06:21
3	shejigao.txt	lisi	增加了许可协议	2019-12-19 21:07:28
4	shejigao.txt	zhangsan	修改版权信息	2019-12-19 21:11:21

1.2 Git与Github

1.2.1 两者区别

Git是一个分布式版本控制系统，简单的说其就是一个软件，用于记录一个或若干个文件内容变化，以便将来查阅特定版本修订情况的软件。

Github (<https://www.github.com>) 是一个为用户提供Git服务的网站，简单说就是一个可以放代码的地方（不过可以放的当然不仅是代码）。Github除了提供管理Git的web界面外，还提供了订阅、关注、讨论组、在线编辑器等丰富的功能。GitHub被称为**全球最大的基友网站**。

###1.2.2 GitHub注册

打开GitHub官网：<https://www.github.com>，点击右上角的“sign up”按钮

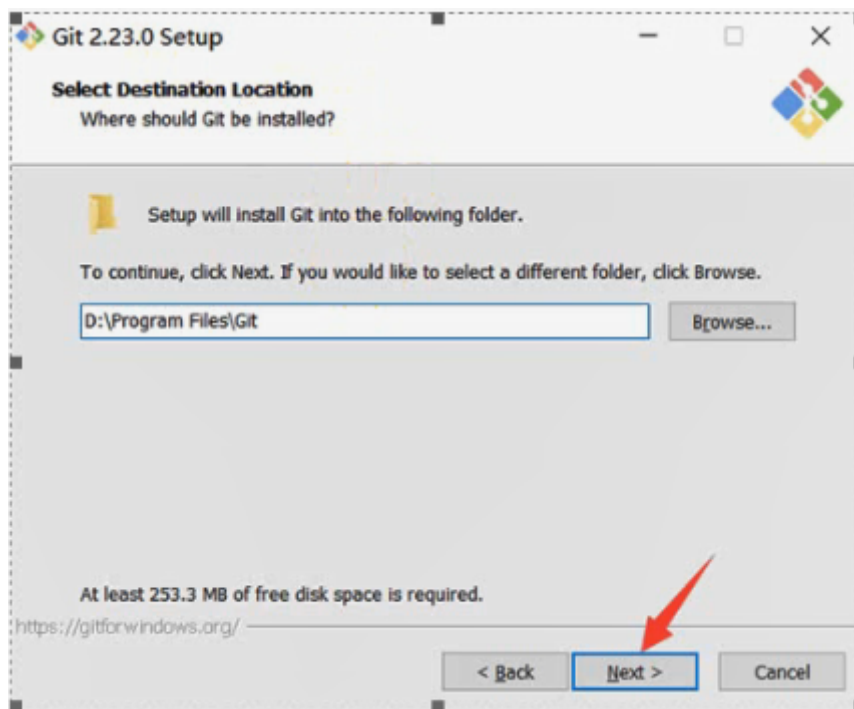
选择免费的账户类型

1.3 Git安装

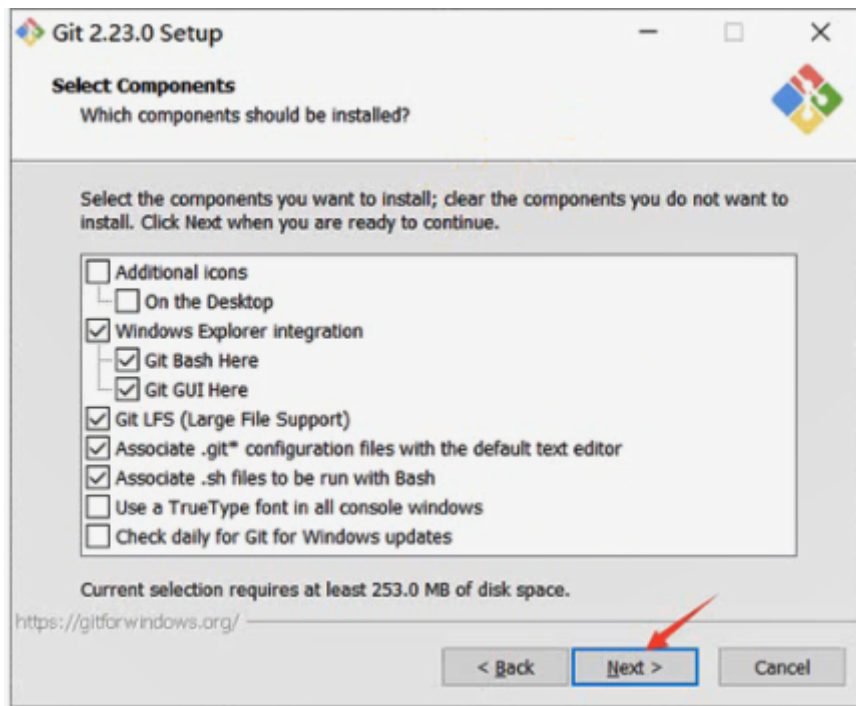
①下载得到安装包，并运行

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

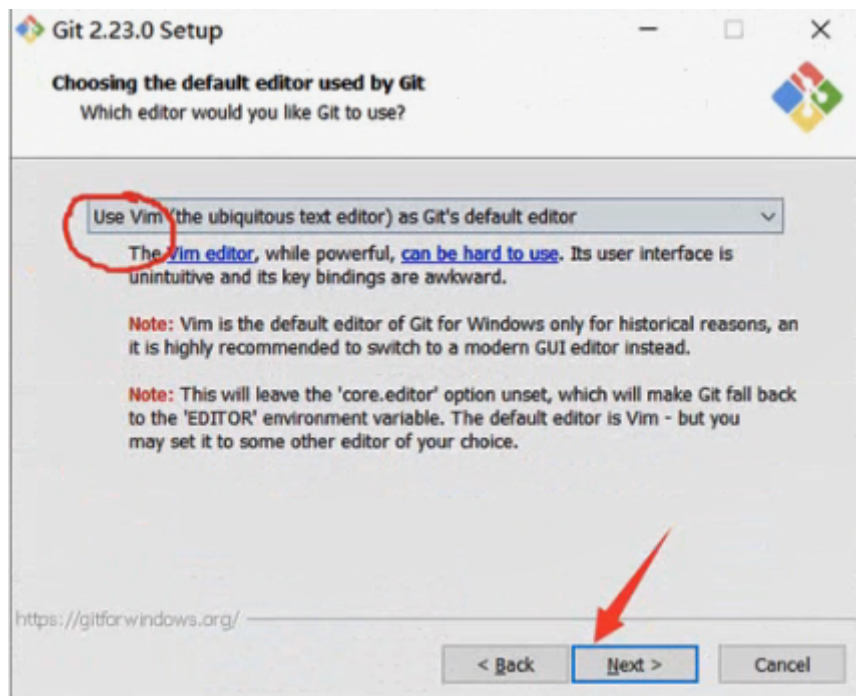
②选择软件的安装位置（建议非中文，非空格目录）



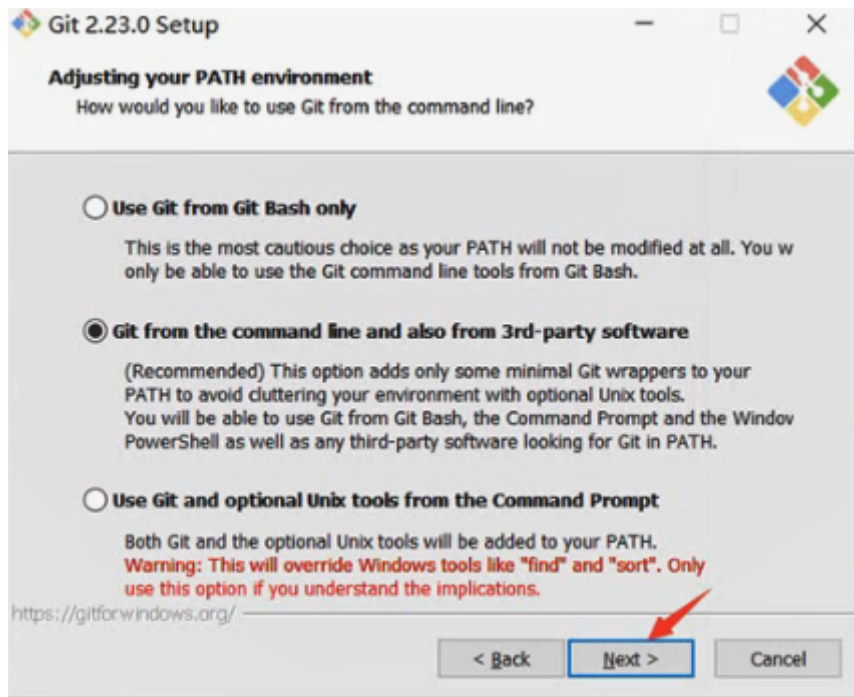
③选择需要安装的组件（默认即可，直接下一步）



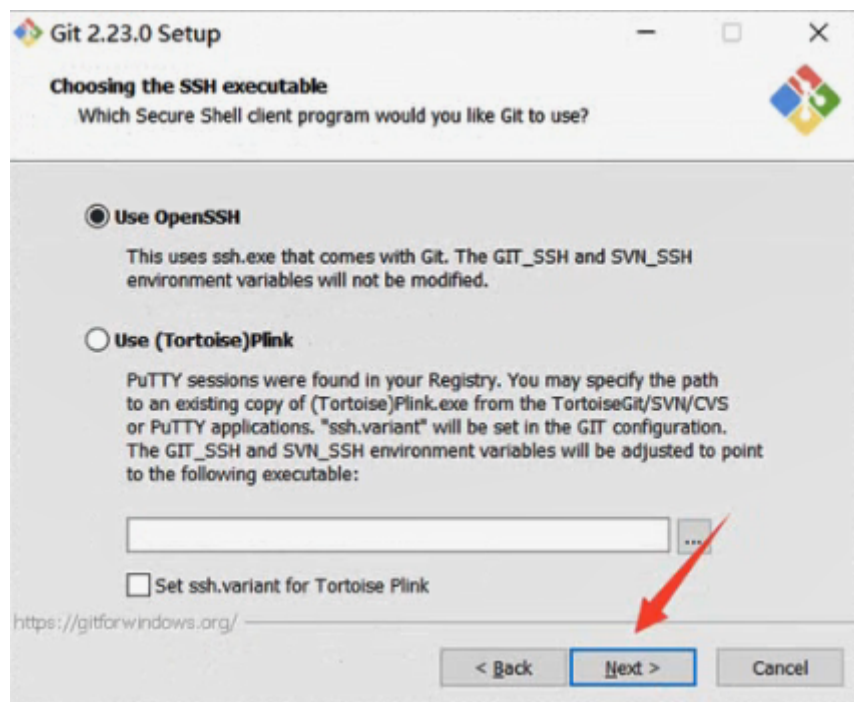
④选择使用的编辑器（默认即可，直接下一步）



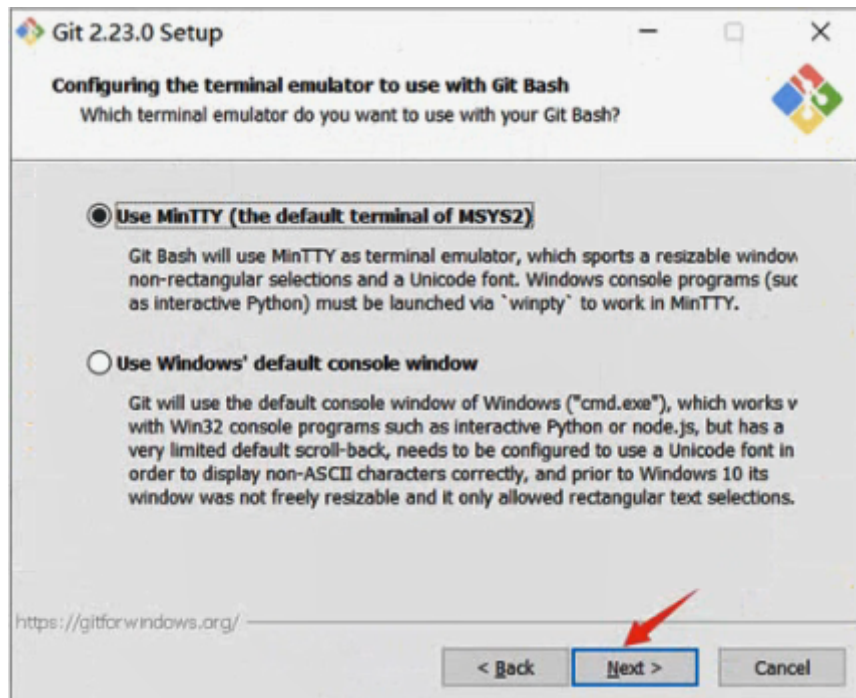
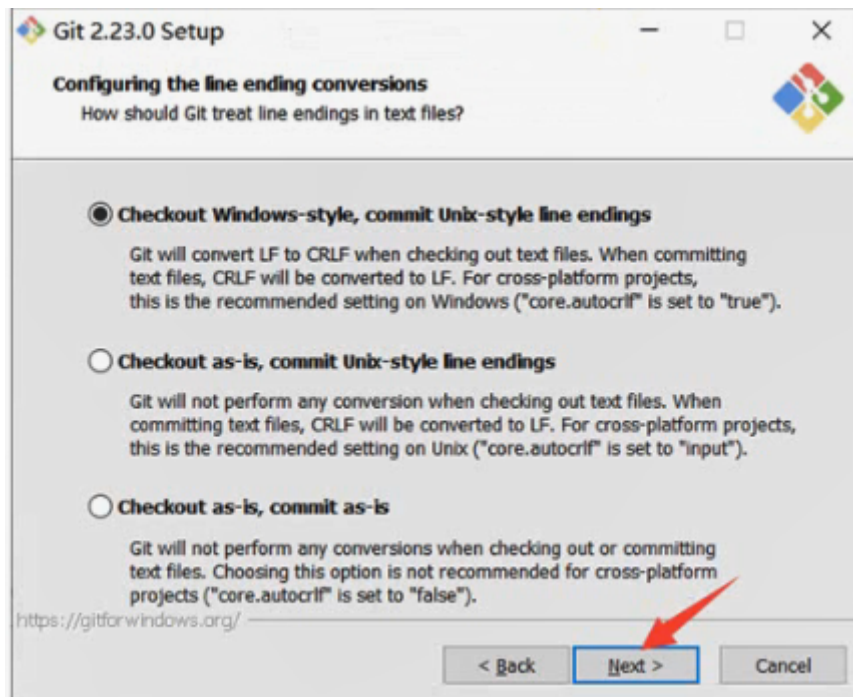
⑤默认即可，直接下一步

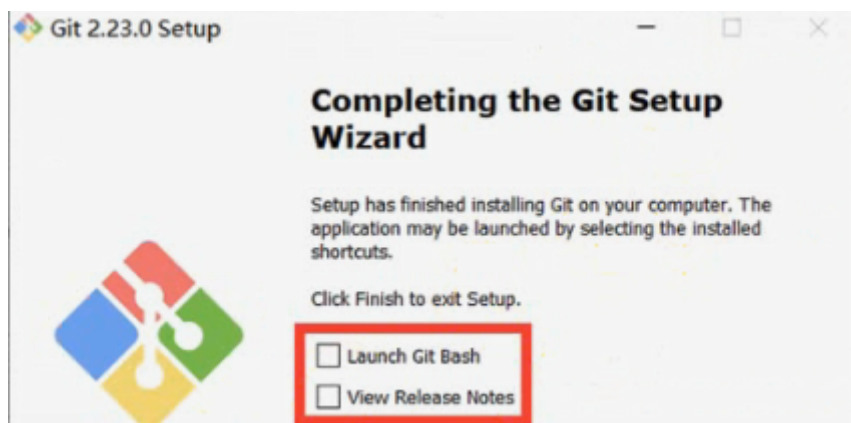
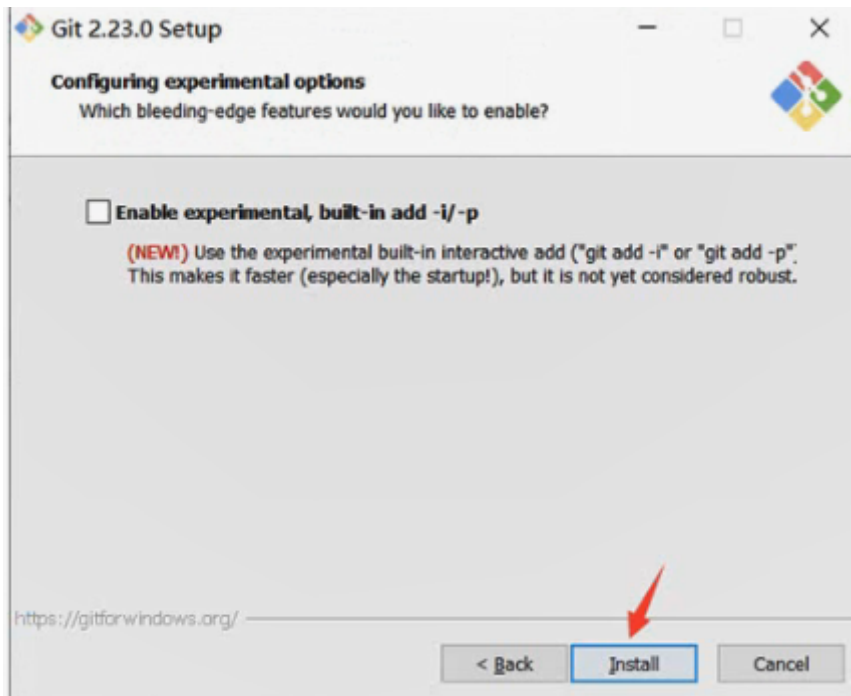
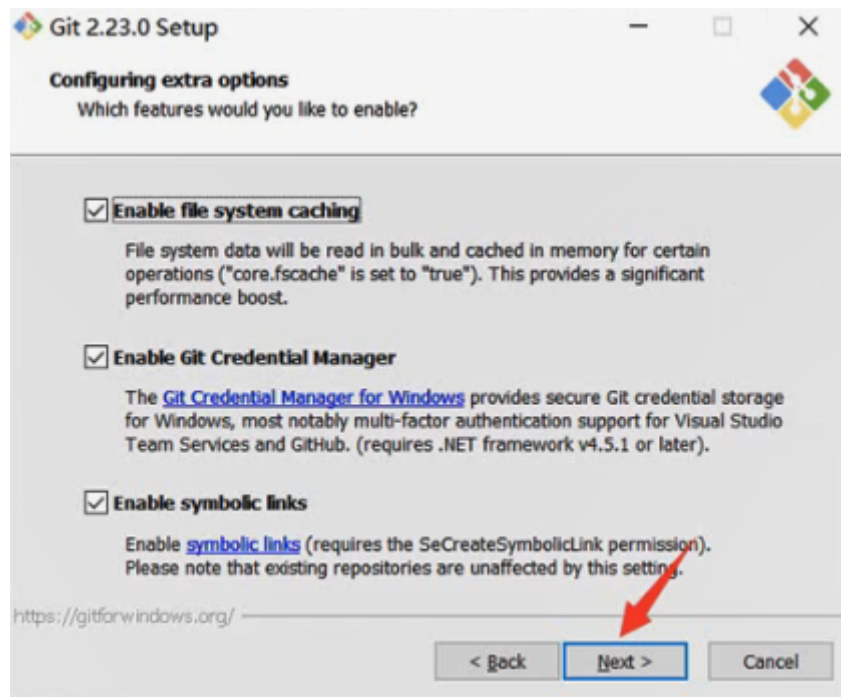


⑥使用Open SSH，直接下一步即可



⑦



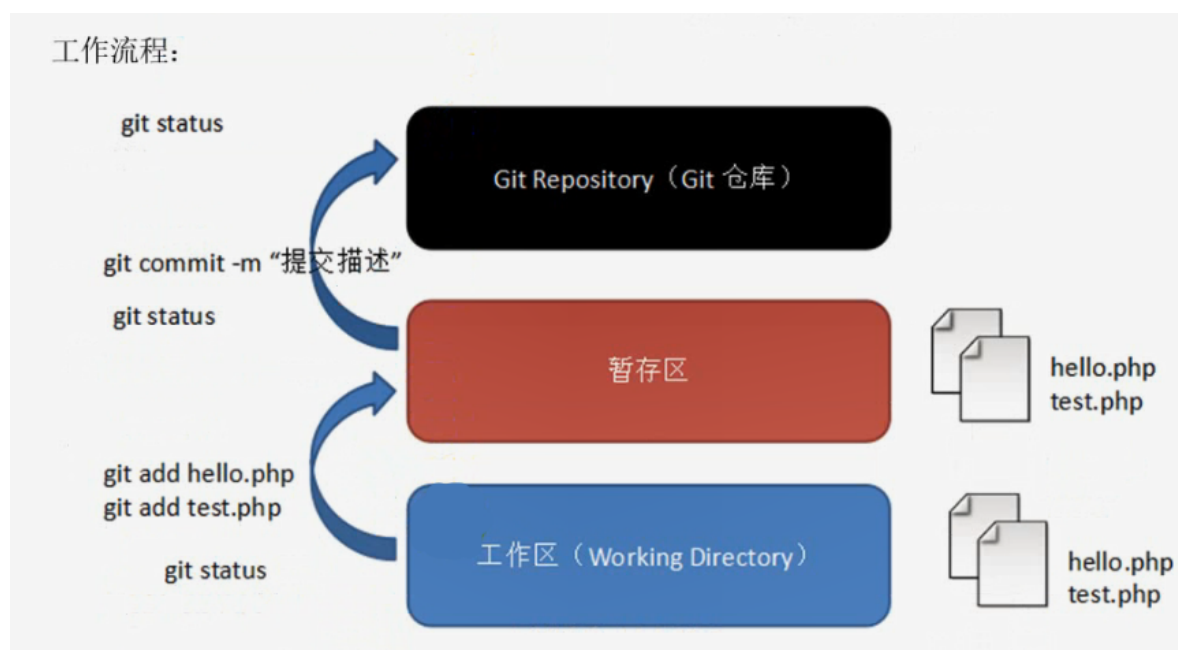


2. Git使用

2.1 本地仓库

2.1.1 工作流程

Git本地操作的三个区域：



2.1.2 本地仓库操作

什么是仓库？仓库又名版本库，英文名Repository，我们可以简单理解成是一个目录，用于存放代码的，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、删除等操作Git都能跟踪到。

①在安装好后首次使用需要先进行全局配置

桌面空白处右键，点击“Git Bash Here”以打开Git命令行窗口

```
1 $git config --global user.name "用户名"
2 $git config --global user.email "邮箱地址"
```

②创建仓库

当我们需要让Git去管理某个新项目/已存在项目的时候，就需要创建仓库了。注意：创建仓库时使用的目录不一定要求是空目录，选择一个非空目录也是可以的，**但是不建议在现有项目上来学习Git，否则造成的一切后果概不负责！**

注意：为了避免在学习或使用过程中出现各种奇葩问题，请不要使用包含中文的目录名（父目录亦是如此）

a. 创建空目录

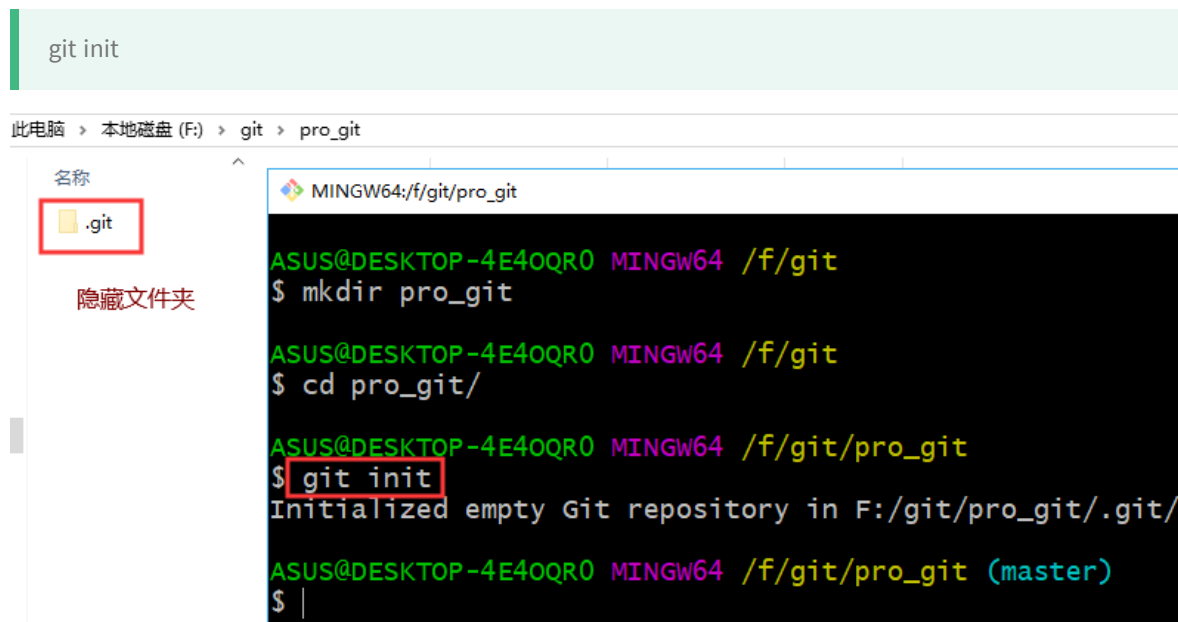


b. 在命令行中进入项目目录pro_git

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git
$ cd pro_git/

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git
$
```

c. Git仓库初始化（让Git知道，它需要来管理整个目录）



表现：执行之后会在项目目录下创建 “.git” 的隐藏目录，这个目录是Git所创建的，不能删除，也不能随意更改其中的内容。

③Git常用指令操作

查看当前状态： `git status` 【非必要】

添加到缓存区： `git add 文件名`

说明：git add指令，可以添加一个文件，也可以同时添加多个文件

语法1：git add 文件名

语法2：git add 文件名1 文件名2 文件名3

语法3：git add .【添加当前目录到缓存区中】

提交至版本库： `git commit-m "注释内容"`

在后续对于文件（可以操作1个或多个）操作之后，重复使用git add 与git commit指令即可。

2.1.3 时光穿梭机——版本回退

版本问题分两步骤进行操作：

①查看版本，确定需要回到的时刻点

指令： `git log` `git log --pretty=oneline`，推荐使用方式2

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   a.txt
        deleted:    b.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git commit -m "add a.txt"
[master 95e1208] add a.txt
2 files changed, 1 insertion(+), 2 deletions(-)
delete mode 100644 b.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ |
```

②回退操作： `git reset --hard 提交编号`

案例：想坐时光机回到创建好第一个文件a.txt的时候

```
此电脑 > 本地磁盘 (F:) > git > pro_git

名称
a.txt

MINGW64:/f/git/pro_git

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git log --pretty=oneline
c4de96e7dd19199f7e67b3251d97bff3324afb46 (HEAD -> master) add b.txt
95e1208a927d321f67afbfe8d6b81003c043b5eb add a.txt
dfcef6175db6947c216ef88b233ddac9d6fc9f7d add a.txt
d54bfb23c12607baf284a0371fa256864c7fd85c add b.txt
2506022da2e9b5b8963a13834c87c4815377cd49 add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git reset --hard 95e1208a927d321f67afbfe8d6b81003c043b5eb
HEAD is now at 95e1208 add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$
```

注意：回到过去之后，要想再回到之前最新的版本的时候，则需要使用指令去查看历史操作，以得到最新的commit id。 `git reflog`

```
此电脑 > 本地磁盘 (F:) > git > pro_git

名称
a.txt
b.txt

MINGW64:/f/git/pro_git

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git reset --hard 95e1208a927d321f67afbfe8d6b81003c043b5eb
HEAD is now at 95e1208 add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git reflog
95e1208 (HEAD -> master) HEAD@{0}: reset: moving to 95e1208a927d321f67afbfe8d6b81003c043b5eb
c4de96e HEAD@{1}: commit: add b.txt
95e1208 (HEAD -> master) HEAD@{2}: commit: add a.txt
dfcef61 HEAD@{3}: commit: add a.txt
d54bfb2 HEAD@{4}: commit: add b.txt
2506022 HEAD@{5}: commit (initial): add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git reset --hard 95e1208
HEAD is now at 95e1208 add a.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$ git reset --hard c4de96e
HEAD is now at c4de96e add b.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/pro_git (master)
$
```

小结：

1. 要想回到过去，必须先得到commit id，然后通过 `git reset --hard` 进行回退；
2. 要想回到未来，需要使用 `git reflog` 进行历史操作查看，得到最新的commit id
3. 在写回退指令的时候，commit id可以不用写全，git自动识别，但是也不能写太少，至少需要写前4位字符

2.2 远程仓库

线上仓库的操作学习以GitHub为例

2.1 线上仓库创建

Owner **Repository name ***

zaz5630 /

Great repository names are short and memorable. Need inspiration? How about **animated-octo-eureka**?

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

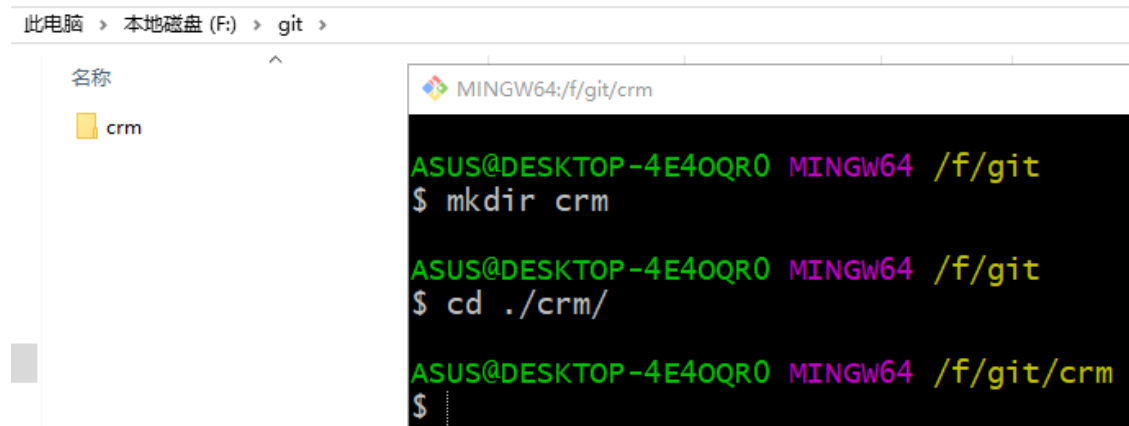
Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

2.2.2 两种常规使用方式

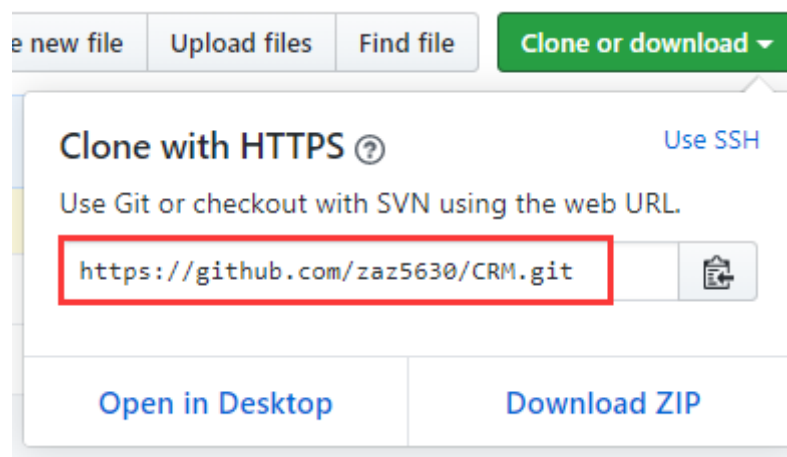
1. 基于Http协议

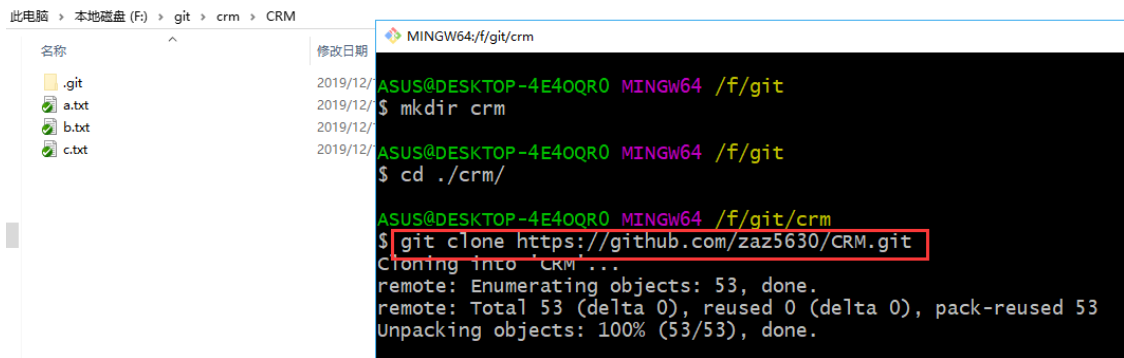
a. 创建空目录，名称为crm



b. 使用 `clone` 指令克隆线上仓库到本地

语法: `git clone` 线上仓库地址





c.在仓库上做对应的操作（提交暂存区、提交本地仓库、提交线上仓库、拉取线上）

提交到线上仓库的指令：`git push`

注意：在首次往线上仓库crm提交内容的时候若出现了404的致命错误，原因是不是任何人都可以往线上仓库提交内容，必须需鉴权。

需要修改“.git/config”文件

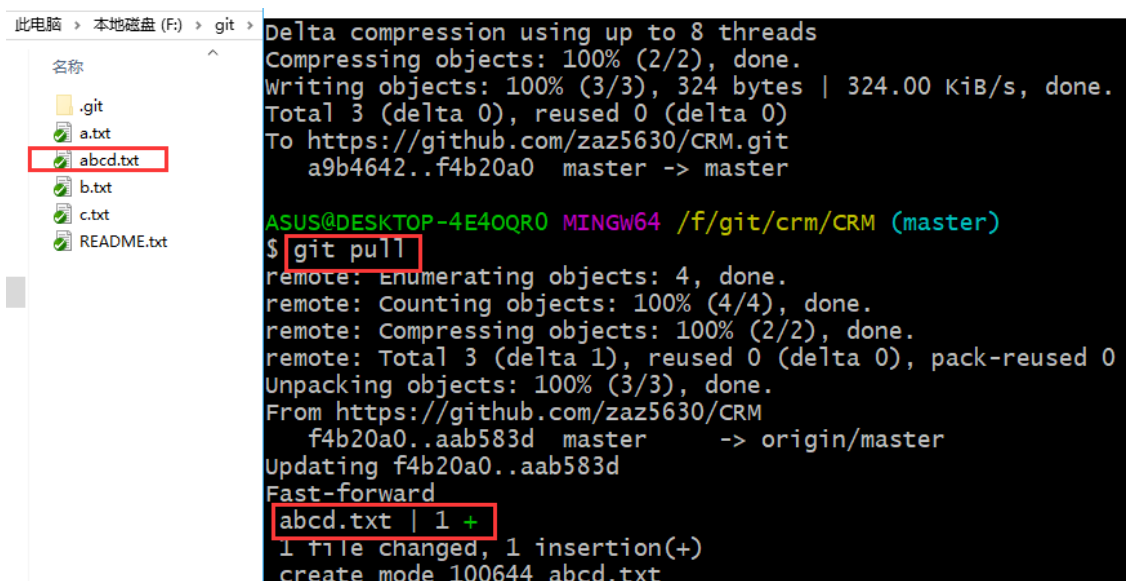
```
1 url = https://github.com/zaz5630/CRM.git
2 修改为:
3 url = https://用户名:密码@github.com/zaz5630/CRM.git
```

在设置好用户名密码之后再次尝试push指令：

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/crm/CRM (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 324 bytes | 324.00 KiB/s, done
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/zaz5630/CRM.git
a9b4642..f4b20a0 master -> master
```

如果看到类似上述效果（没有fatal错误），则表示提交成功

拉取线上仓库：`git pull`



提醒：

1. 每天工作的第一件事就是先 `git pull` 拉取线上最新的版本；
2. 每天下班前要做的是 `git push`，将本地代码提交到线上仓库

2. 基于SSH协议

该方式与前面https方式相比，只是影响github对于用户的身份鉴权方式，对于git的具体操作（如提交本地、添加注释、提交远程等操作）没有任何影响。

生成公私钥指令（需要先自行安装OpenSSH）：`ssh-keygen -t rsa -C "注册邮箱"`

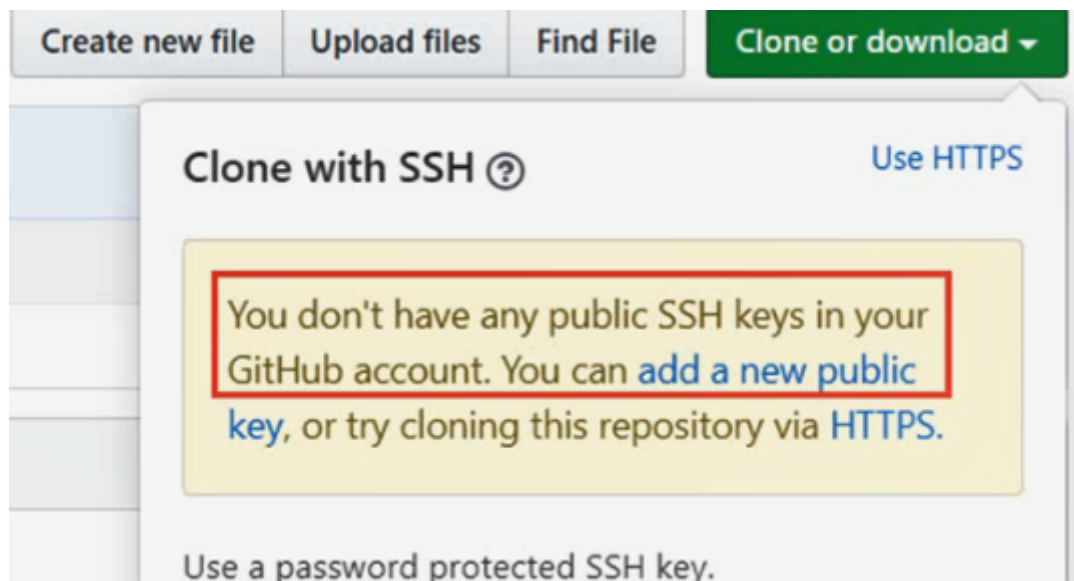
步骤：

①生成客户端公私钥文件

②将公钥上传到Github

实际操作：

①打开提示



②创建公私钥对文件

```
admin@MINGW64 ~/Desktop/shop/shop (master)
$ ssh-keygen -t rsa -C "itcast@cherish.pw"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/admin/.ssh/id_rsa):
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:
SHA256:1/WyDY81+l1REKo5SZiPwuaw2Gk2zvelusAX/qJEkqu itcast@cherish.pw
The key's randomart image is:
+---[RSA 3072]-----+
  o
 . o . . .
+ . + = . .
E o = S B . .oo
* B o . . .0o
 . @ + . .o +
* +o. o . o
+oo=+ o
+---[SHA256]-----+
```

执行指令之后连续回车即可

③上传公钥文件内容（id_rsa.pub）

Personal settings

SSH keys / Add new

Title

Key

Begins with 'ssh-ed25519', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Add SSH key

填写完毕之后保存即可。

④执行后续git操作，操作与先前一样

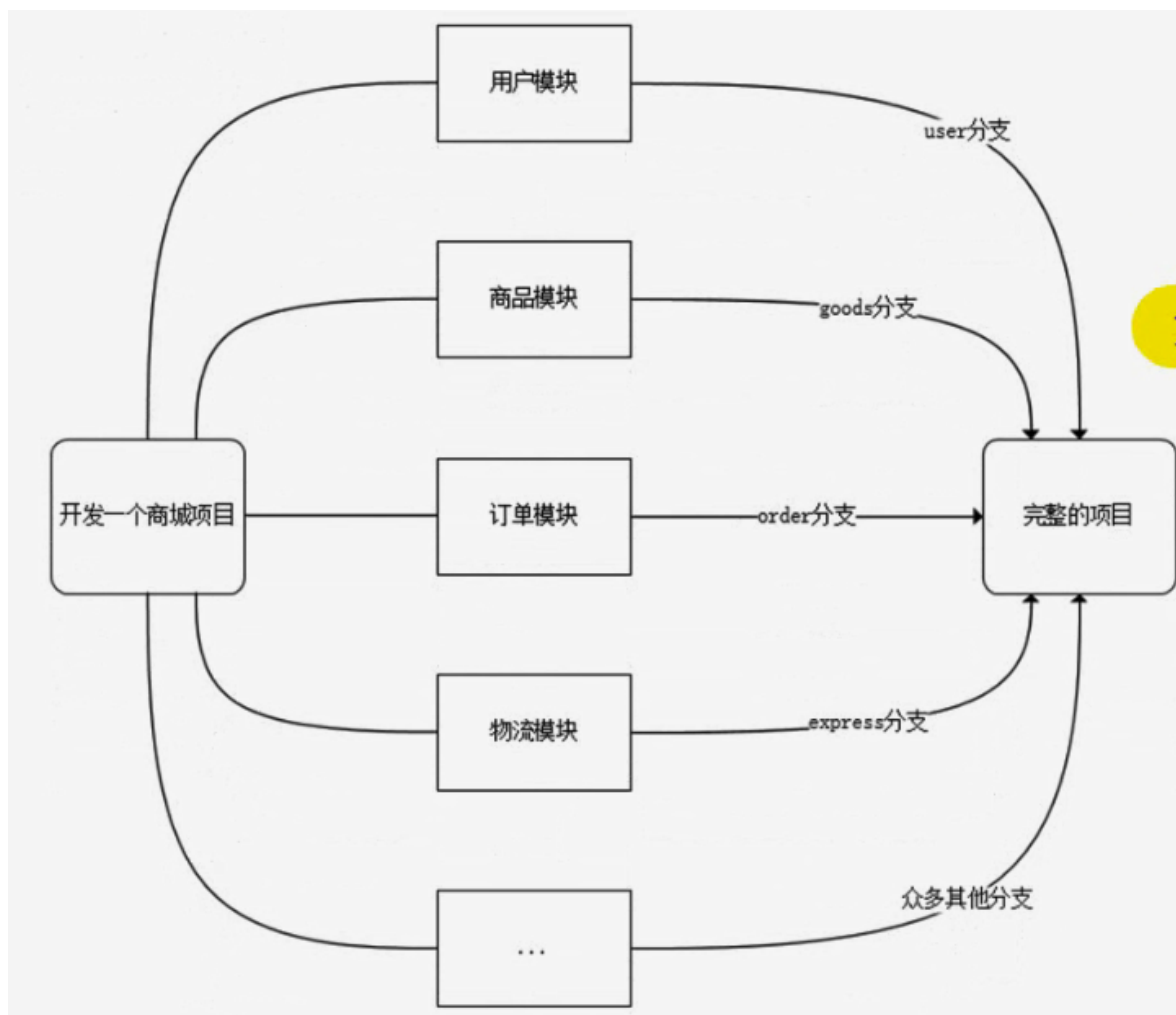
a.clone线上仓库到本地 (`git clone`)

```
此电脑 > 本地磁盘 (F:) > git > withssh >
名称 修改日期 类型 大小
CRM
MINGW64:/f/git/withssh
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh
$ git clone git@github.com:zaz5630/CRM.git
Cloning into 'CRM'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 59 (delta 1), reused 2 (delta 0), pack-reused 53
Receiving objects: 100% (59/59), 5.86 KiB | 749.00 KiB/s, done.
Resolving deltas: 100% (8/8), done.
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh
$ |
```

b.修改文件后添加到缓存区

2.3 分支管理

什么是分支？



在版本回退的章节里，每次提交后都会有记录，Git把他们串成时间线，形成类似于时间轴的东西。这个时间轴就是一个分支，我们称之为 **master分支**。

在开发的时候往往是团队协作，多人进行开发，因此光有一个分支是无法满足多人同时开发的需求的，并且在分支上工作并不影响其他分支的正常使用，会更加安全，Git鼓励开发者使用分支去完成一些开发任务。

分支相关指令：

查看分支：`git branch`

创建分支：`git branch 分支名`

切换分支：`git checkout 分支名`

删除分支：`git branch -d 分支名`

合并分支：`git merge 被合并的分支名`

对于新分支，可以使用 `git checkout -b 分支名` 指令来切换分支，-b选项表示创建并切换，相当于是两个操作指令。

查看分支

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git branch
* master
```

注意：当前分支前面有个标记"*"

创建分支

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git branch dev

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git branch
dev
* master
```


切换分支

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git checkout dev
Switched to branch 'dev'

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (dev)
$ git branch
* dev
master
```

合并分支


先在dev分支下的README.txt文件中新增一行并提交到本地



```
master
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (dev)
$ git add README.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (dev)
$ git commit -m "在readme.txt文件中新增一行"
[dev e085629] 在readme.txt文件中新增一行
1 file changed, 2 insertions(+), 1 deletion(-)
```

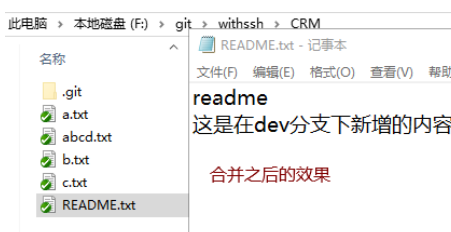
切换到master分支下观察README.txt文件



```
$
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$
```

将dev分支的内容与master分支合并：



```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git merge dev
Updating aab583d..e085629
Fast-forward
 README.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
```

删除分支：

```

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/with
$ git branch -d dev
Deleted branch dev (was e085629).

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/with
$ git branch
* master

```

注意：在删除分支的时候，一定要先退出要删除的分支，然后才能删除。

合并所有分支之后，需要将master分支提交到远程仓库中。

Branch: master CRM / README.txt

zaz5630 在readme.txt文件中新增一行

1 contributor

2 lines (2 sloc) | 32 Bytes

```

1  readme
2  这是在dev分支下新增的内容

```

```

Deleted branch dev (was e085629).

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git branch
* master

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 420 bytes | 420.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:zaz5630/CRM.git
 aab583d..e085629 master -> master

```

2.4 冲突的产生与解决

案例：模拟产生冲突

①同事在下班之后修改了线上仓库的代码

<> Edit file
👁 Preview changes

```

1  readme
2  这是在dev分支下新增的内容
3
4  这是我同事小A在我下班之后做的修改

```

注意：此时我本地仓库的内容与线上不一致的。

CRM /

README.txt

Cancel

<> Edit file
👁 Preview changes

```

1  readme
2  这是在dev分支下新增的内容
3
4  这是我同事小A在我下班之后做的修改

```

README.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

readme

这是在dev分支下新增的内容

②第二天上班的时候，我没有做git pull操作，而是直接修改了本地的对应文件的内容

README.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

readme

这是在dev分支下新增的内容

这些文字是我次日上班的时候写的

③需要在下班的时候将代码修改提交到线上仓库 (git push)

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git push
To github.com:zaz5630/CRM.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:zaz5630/CRM.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

提示我们要在再次push之前先git pull操作。

【解决冲突】

④先git pull

```
ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:zaz5630/CRM
  4a73d76..11c6b65  master    -> origin/master
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
Automatic merge failed; fix conflicts and then commit the result.
```

此时git已经将线上与本地仓库的冲突合并到了对应的文件中

⑤打开冲突文件，解决冲突

解决方法：需要和同事（谁先提交的）进行商量，看代码如何保留，将改好的文件再次提交即可

<<<<<<< HEAD

这些文字是我次日上班的时候写的

=====

这是我同事小A在我下班之后做的修改

>>>>>> 11c6b655e6174288a4db016beb4805c45b8b088a

将内容进行调整，保留需要的，不
需要的删除即可

⑥重新提交

```

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master|MERGING)
$ git add README.txt

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master|MERGING)
$ git commit -m "解决了冲突"
[master 8a24c0b] 解决了冲突

ASUS@DESKTOP-4E4OQR0 MINGW64 /f/git/withssh/CRM (master)
$ git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 785 bytes | 392.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:zaz5630/CRM.git
   11c6b65..8a24c0b  master -> master

```

线上效果


zaz5630 解决了冲突

1 contributor

10 lines (5 sloc) | 194 Bytes

```

1  readme
2  这是在dev分支下新增的内容
3
4  这些文字是我次日上班的时候写的
5
6
7  这些文字是我次日上班的时候写的
8
9  这是我同事小A在我下班之后做的修改

```

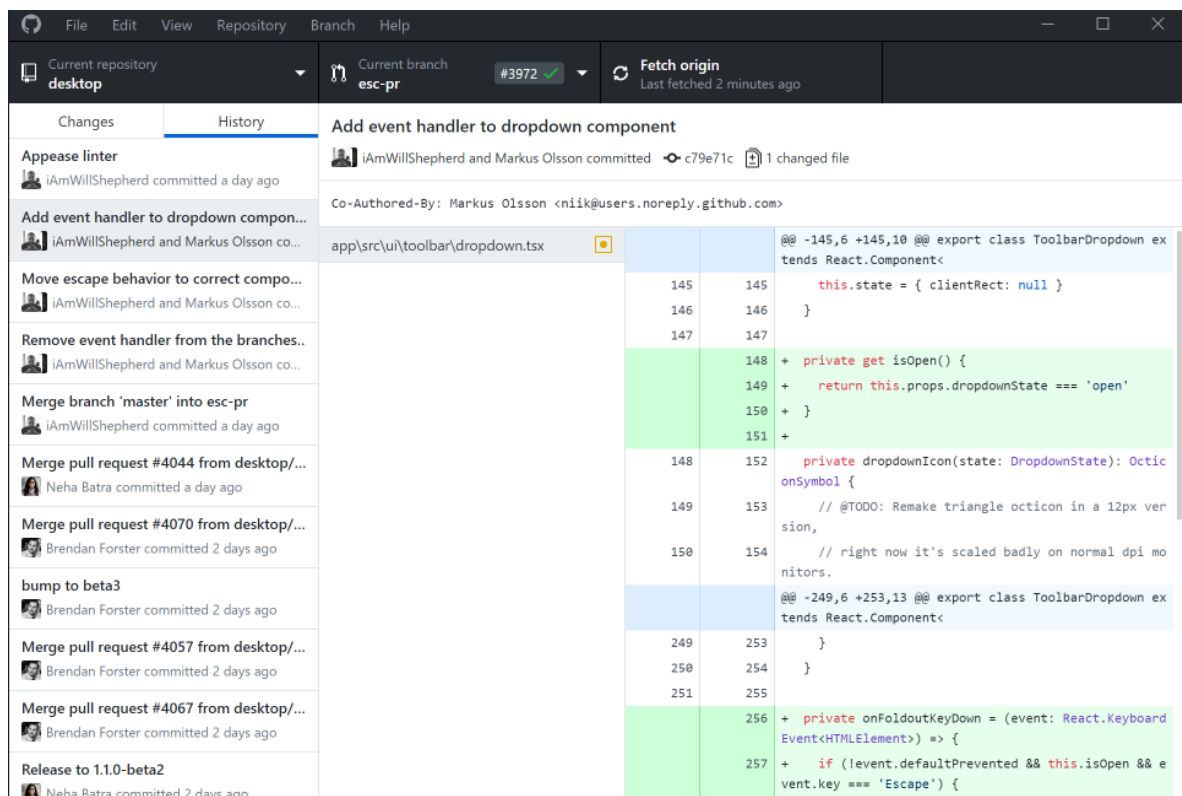
此时线上是最新

3. Git实用技能

3.1 图形管理工具

①GitHub for Desktop

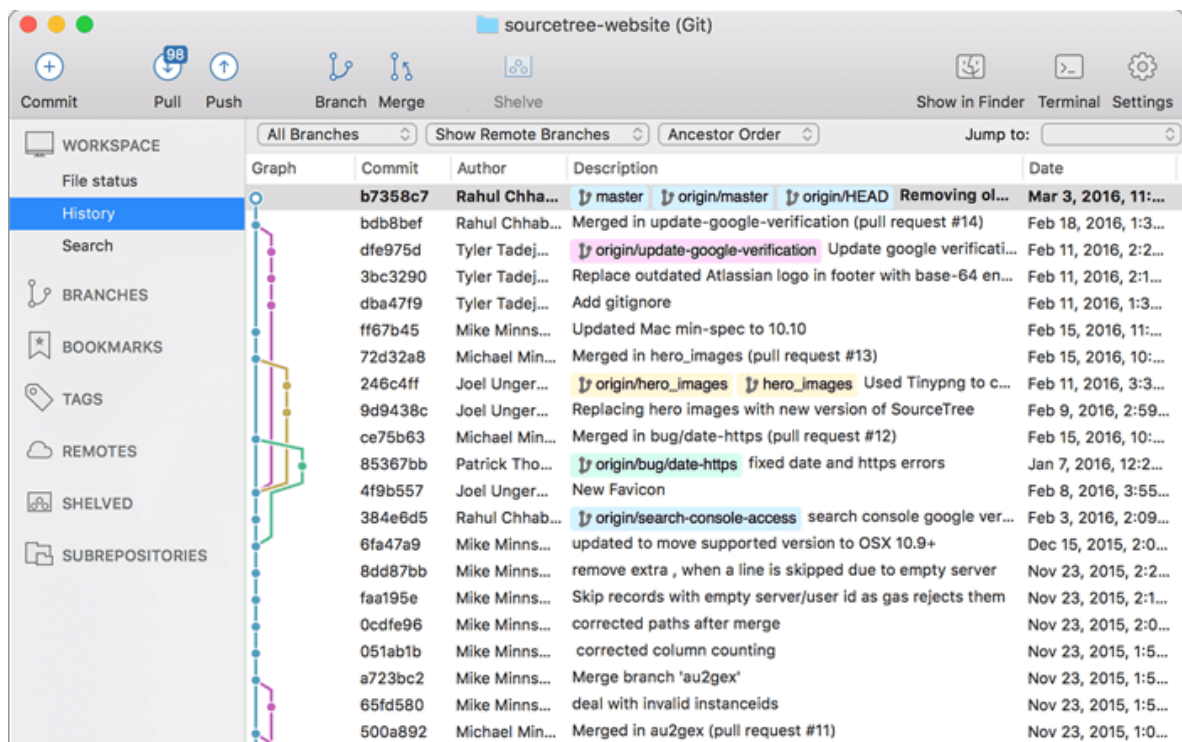
<https://desktop.github.com/>



GitHub出品的软件，功能完善，使用方便。对于经常使用GitHub的开发人员来说是非常便捷的工具。界面干净，用起来非常顺手，顶部的分支时间线非常绚丽。

②Source Tree

<https://www.sourcetreeapp.com/>



老牌的Git GUI管理工具，也号称是最好用的Git GUI工具，功能丰富，基本操作和高级操作都非常流程，适合初学者上手。

③TortoiseGit

<https://tortoisegit.org/download/>



对于熟悉SVN的开发人员来说，这个小乌龟图标应该是非常友善了。TortoiseGit简称tgit，中文名海龟Git。它与其前辈TortoiseSVN都是非常优秀的开源版本控制客户端软件。

3.2 忽略文件

场景：在项目目录下有很多万年不变的文件目录，例如css、js、image等，或者还有一些目录即便有改动，我们也不想让其提交到远程仓库的文档，此时我们可以使用“忽略文件”机制来实现需求。

忽略文件需要新建一个名为 `.gitignore` 的文件，该文件用于声明忽略文件或不忽略文件的规则，规则对当前目录及其子目录生效。

注意：该文件因为没有文件名，没办法直接在windows目录下直接创建，可以通过命令Git bash来touch创建

场景规则写法有如下几种：

1. `/mtk/` 过滤整个文件夹
2. `*.zip` 过滤所有.zip文件
3. `/mtk/do.c` 过滤某个具体文件
4. `!index.jsp` 不过滤某个具体文件
5. 在文件中，以#开头的都是注释

案例：

①先在本地仓库中新建一个js目录以及目录中js文件

②依次提交本地到线上

③新增 `.gitignore` 文件

④编写文件中规则

```
1  #忽略掉/js目录
2  /js/
```

⑤再次提交本地到线上，观察线上仓库js目录中是否有对应的js文件