

Git 版本控制工具

课程计划

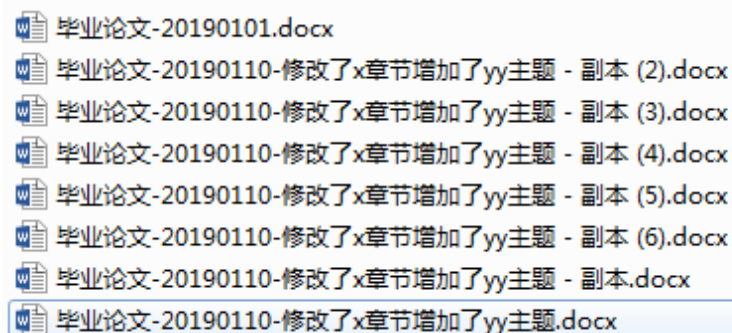
- 什么是版本控制[理解]
- Git 介绍[了解]
- Git 工作流程[了解]
- Git 安装[应用]
- Git 的结构[认识]
- Git 命令行操作进行介绍（已经涵盖后续我们要使用的所有命令）[掌握]
- GitHub（和 Git 什么关系，如何使用）[应用]
- Idea 集成 Git，然后在 Idea 中直接使用界面操作 Git 完成我们的操作[掌握]

一、版本控制[理解]

（一）版本控制思想

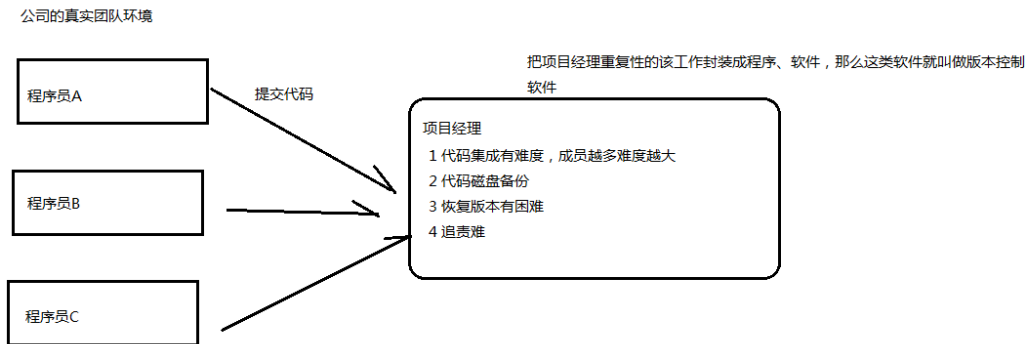
从个人工作和团队协作两个方面（角度）介绍版本控制思想

- 个人工作



往往为了恢复到某一个版本，我们不停的对我们的文件进行复制修改（备份）

- 团队协作



版本从何而来呢？

设定场景：

SVN，123

Git，hash 码

A 提交一次代码，服务器的版本为 1

B 提交一次代码，服务器的版本为 2

...

每提交一次代码，服务器的文件版本+1，这样每次提交的东西都有一个版本号对应，易于管理，所以这种软件就叫做版本控制软件

安装了版本控制软件的服务器叫做，版本控制服务器，作用：团队协作开发。

文件上传到版本控制服务器，仅有文件内容还不够，还需要文件的元数据信息（描述数据的数据就叫做元数据）（修改时间、修改人、备注注释信息等），这样才能够便于恢复版本、追责等

（二）版本控制软件应该具有的功能

- 协同修改
多人并行不悖的修改服务器端的同一个文件。
- 数据备份
不仅保存目录和文件的当前状态， 还能够保存每一个提交过的历史状态。

- 版本管理

在保存每一个版本的文件信息的时候要做到不保存重复数据，以节约存储空间，提高运行效率。这方面 SVN 采用的是增量式管理的方式，而 Git 采取了文件系统快照的方式

- 历史记录

查看修改人、 修改时间、 修改内容、 日志信息。
将本地文件恢复到某一个历史状态。

- 分支管理

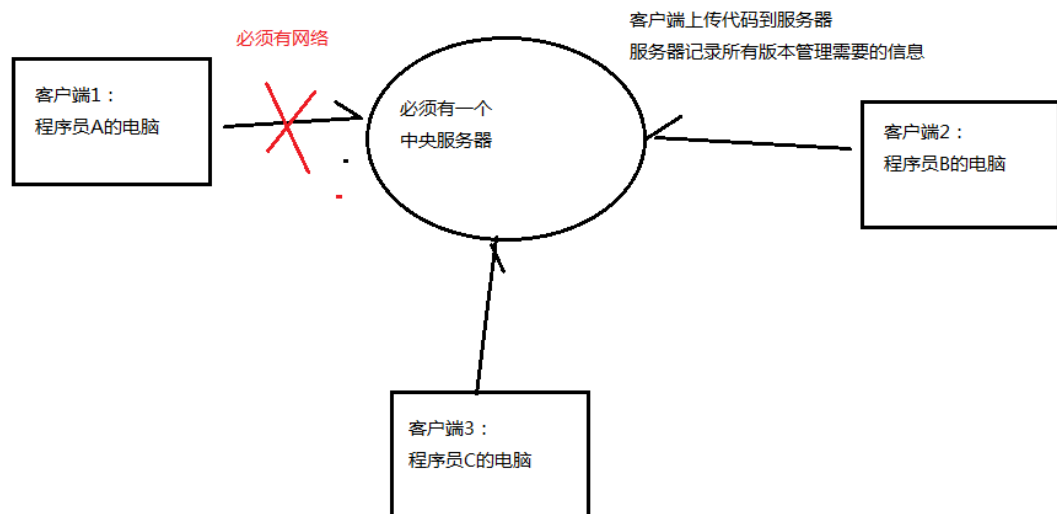
允许开发团队在工作过程中多条生产线同时推进任务， 进一步提高效率。

注意：版本控制不仅适用于 IT 领域

(三) 版本控制工具介绍

- 集中式版本控制工具

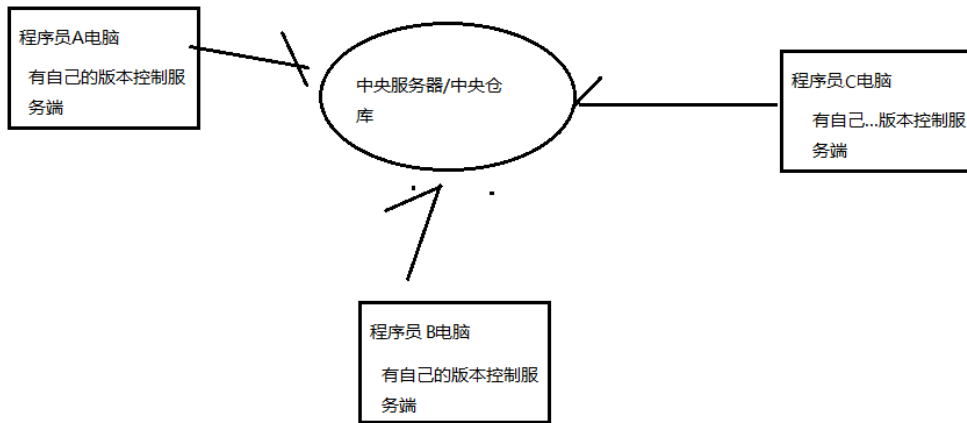
代表：cvs、**svn**、vss



弊端：容错性不好，服务器宕机，磁盘坏掉

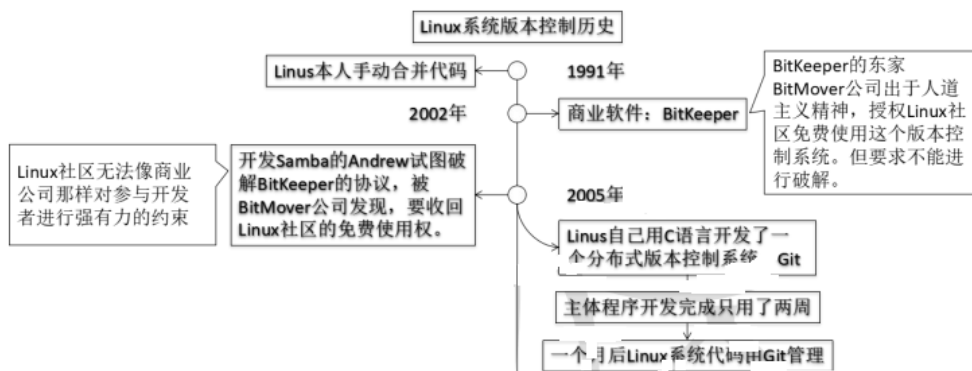
- 分布式版本控制工具

代表性：**git**、BitKeeper



二、Git 介绍[了解]

(一) Git 简史



(二) Git 官网和 Logo

官网地址: <https://git-scm.com/>



(三) Git 的优势

- 大部分操作在本地完成，不需要联网（不折不扣的分布式）
- 完整性保证

Hash 算法

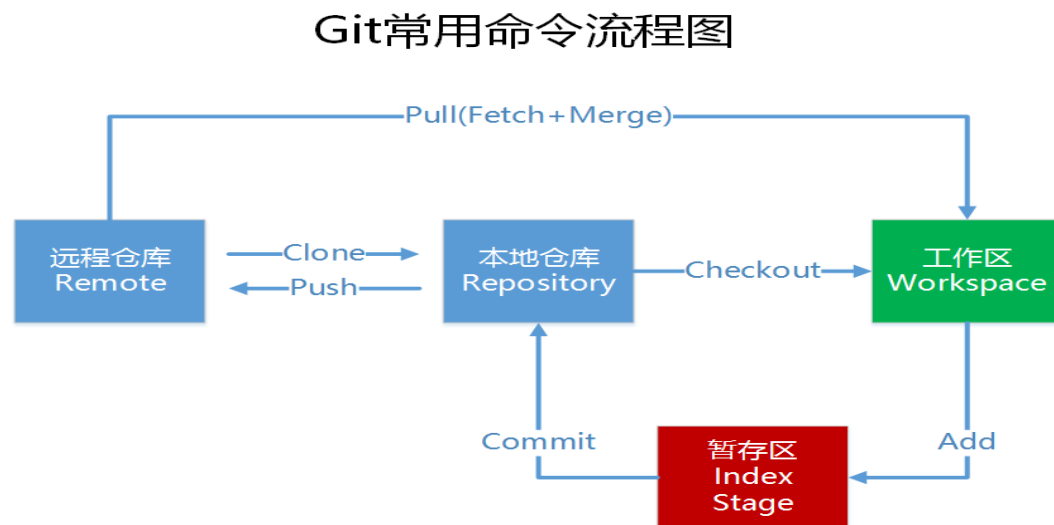
- 尽可能添加数据而不是删除或修改数据
- 分支操作非常快捷流畅
- 与 Linux 命令全面兼容

三、Git 工作流程[了解]

一般工作流程如下：

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：



四、Git 的安装[应用]

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，所以本教程只讲解 windows 下的 git 的安装及使用。

(一) 软件下载

下载地址: <https://git-scm.com/download>



(二) 软件安装[应用]

1. 安装 git for windows



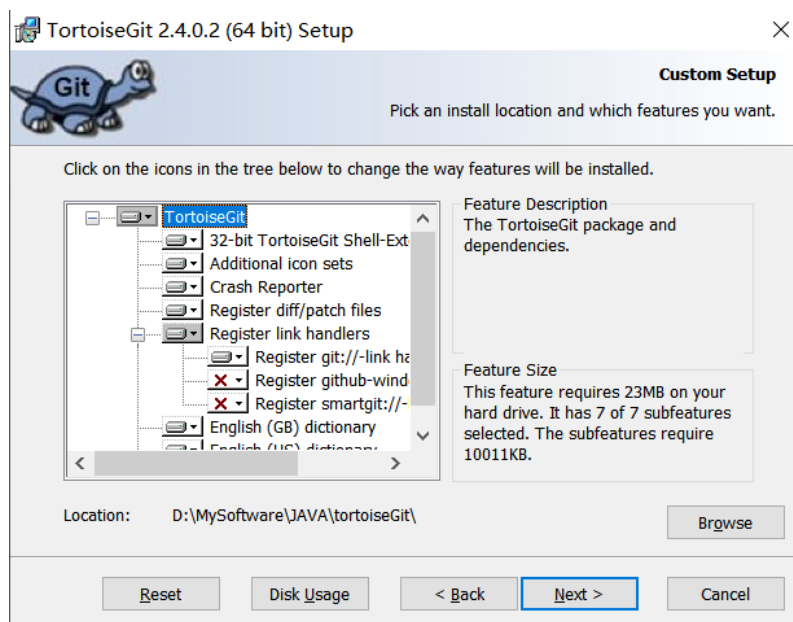
一路“下一步”使用默认选项即可。**注意：安装路径不要用中文**

■ 验证安装是否成功

在任何一个目录下右键看是否有菜单

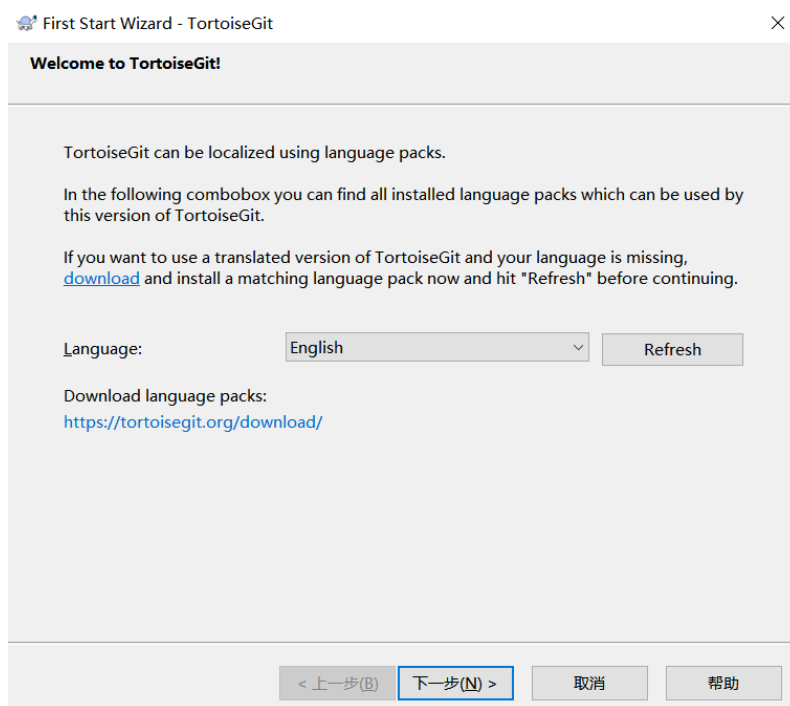


2. 安装 TortoiseGit

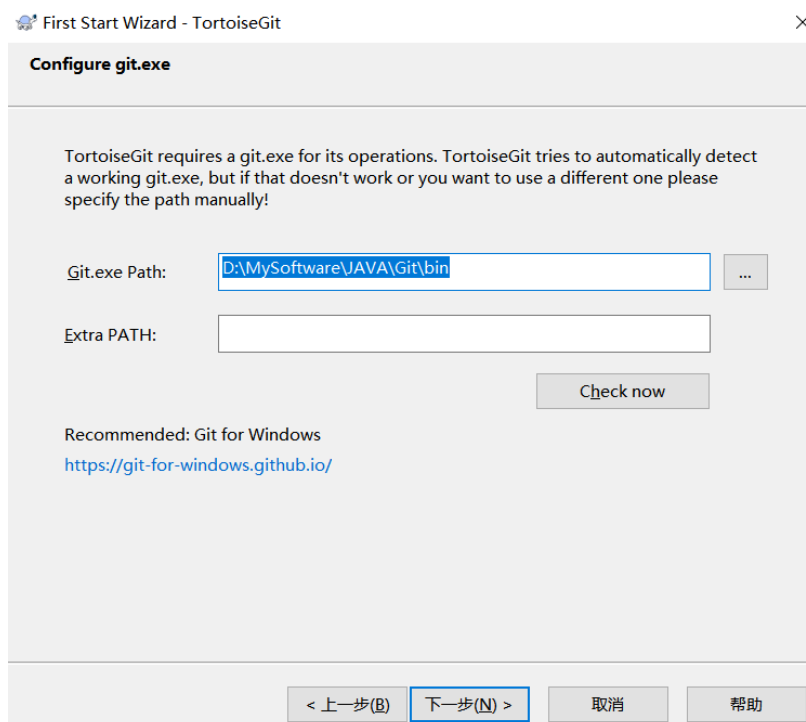


一路“下一步”使用默认选项即可。

默认选项下会启动配置画面：



由于目前只有英文语言包，默认即可继续下一步。



配置 git.exe，在 4.2.1 中已经安装过 git-for-windows 了所以在此找到 git.exe 所在的目录。

First Start Wizard - TortoiseGit

Configure user information

Git requires that you set up a user name and email address. Both are used as meta data for your commits (not for authentication).

Name:

Email:

These settings will be stored to your global git configuration (%HOME%/.gitconfig) and will be used for all your git repositories as a default.

☐ Don't store these settings now.

< 上一步(B) 下一步(N) > 取消 帮助

配置开发者姓名及邮箱，每次提交代码时都会把此信息包含到提交的信息中。

First Start Wizard - TortoiseGit

Authentication and credential store

SSH (URLs look like "git@example.com")

TortoiseGitPlink is recommended as SSH client. If you don't have a key pair yet, you should generate one. Keep the private one in a save place and set up the public key on your hosting platform. Use the PuTTY authentication agent for caching the password (done automatically if a PuTTY key is configured for a remote). For advanced tips & tricks see our [manual](#) and [FAQ](#).

HTTP (URLs start with "http://" or "https://")

By default Git does not save/cache credentials. However, you can configure a credential helper (recommended) or manually use %HOME%/_netrc.

Credential helper:

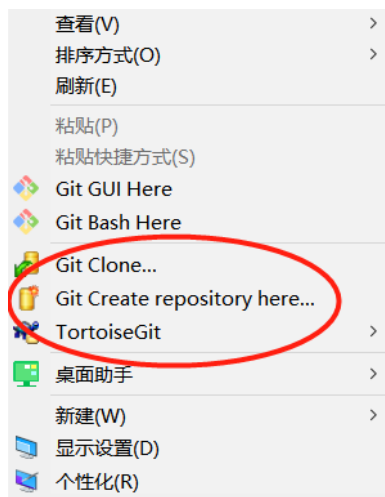
☐ Don't store these settings now.

These settings will be stored to your global git configuration (%HOME%/.gitconfig) and will be used for all your git repositories as a default.

< 上一步(B) 完成 取消 帮助

使用默认配置，点击“完成”按钮完成配置。

完整完毕后在系统右键菜单中会出现 git 的菜单项。



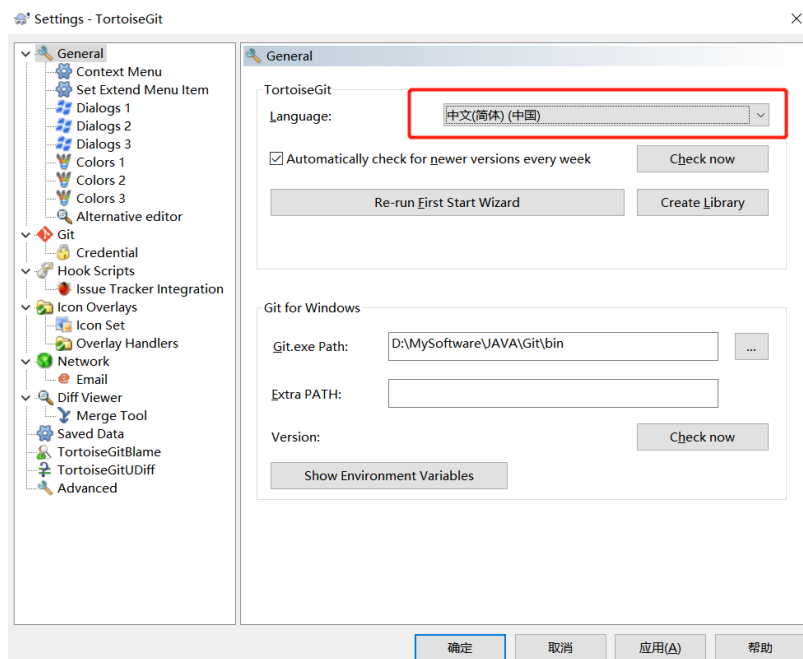
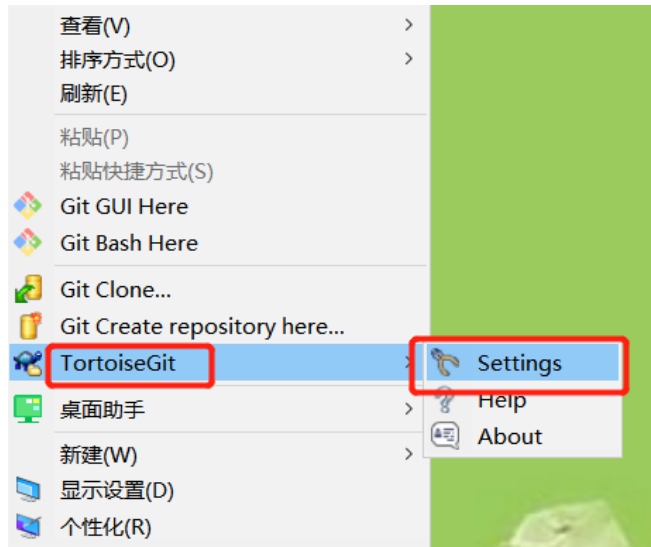
3. 安装中文语言包

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。



直接“下一步”完整完毕。

语言包安装完毕后可以在 TortoiseGit 的设置中调整语言



五、Git 的结构[认识]

(一) Git 的本地结构



(二) Git 的代码托管中心

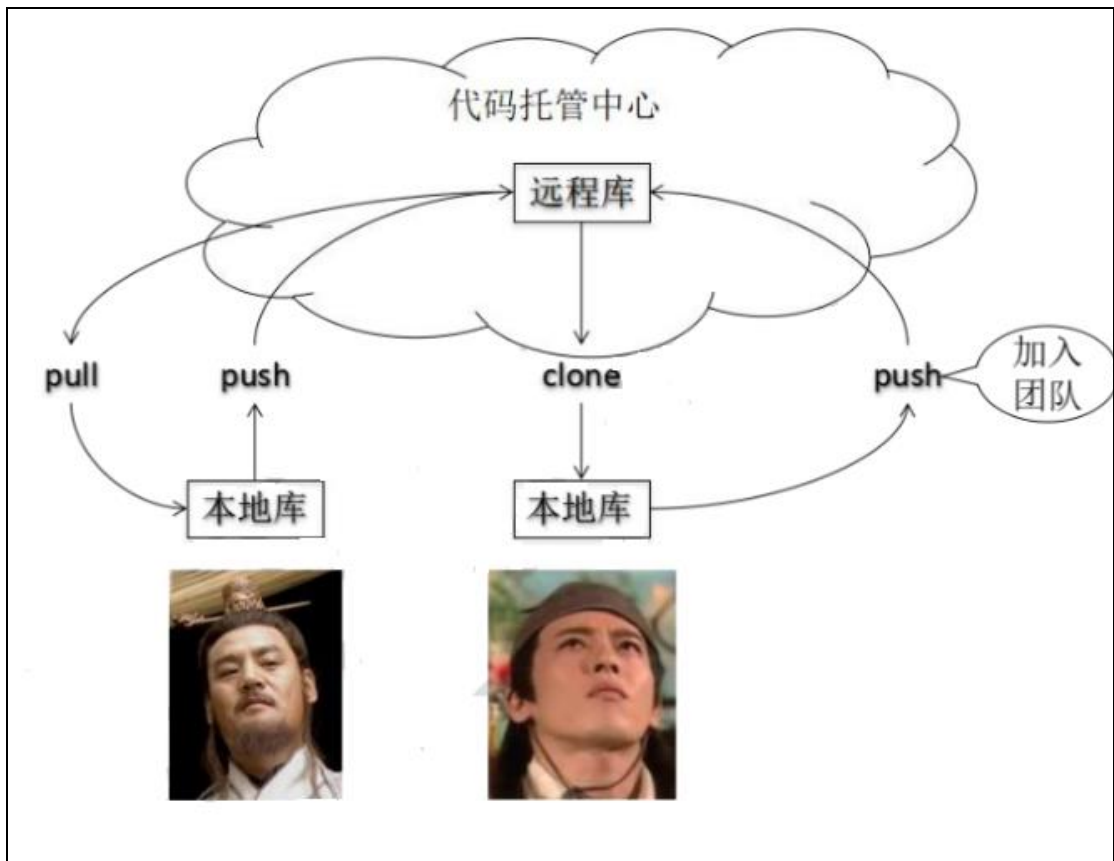
代码托管中心的任务：维护远程库

- 局域网环境下
 - GitLab 服务器
- 外网环境下
 - GitHub
 - 码云

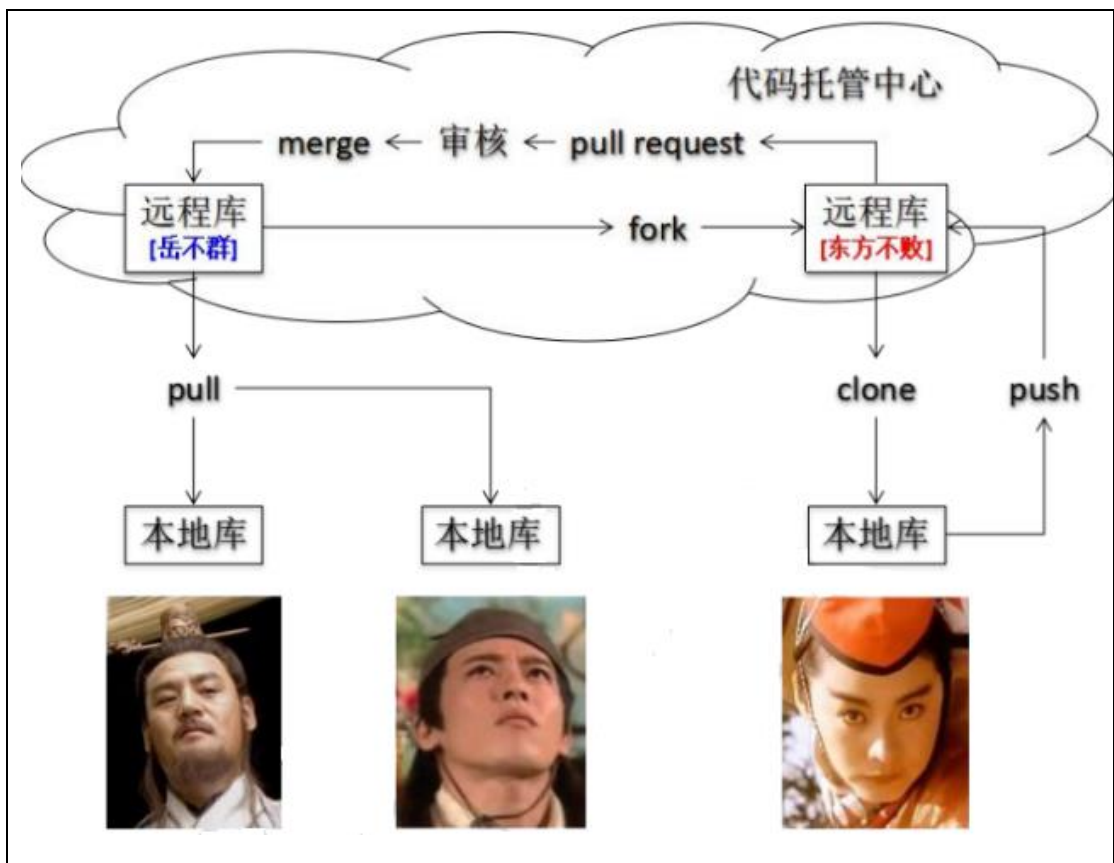
(三) 本地库和远程库

本地库和远程库的交互方式，介绍这么两种：一种是团队内部协作，一种是跨团队协作

- 团队内协作



● 跨团队协作



六、Git 命令行操作[掌握]

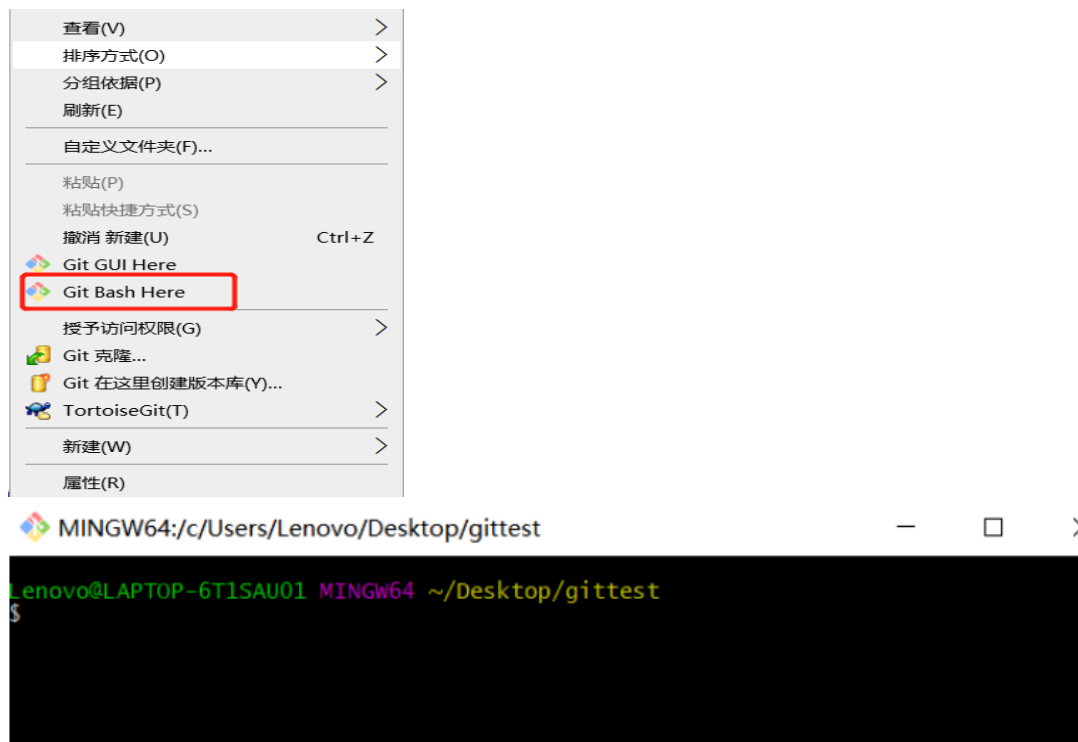
Git 本地项目仓库(可以多个)初始化

命令行操作学习路线：初始化 git 本地版本库——>在本地版本库管理文件（新建文件 ——> 把文件加入 git 的管理 ——> 修改文件 ——> 恢复到之前某一个版本）

（一）创建版本库

1. 使用 GitBash

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：

- 命令：git init
- 效果

```

MINGW64:/c/Users/Lenovo/Desktop/gittest
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest
$ git init
Initialized empty Git repository in C:/Users/Lenovo/Desktop/gittest/.git/
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ |
  
```

此电脑 > 桌面 > gittest

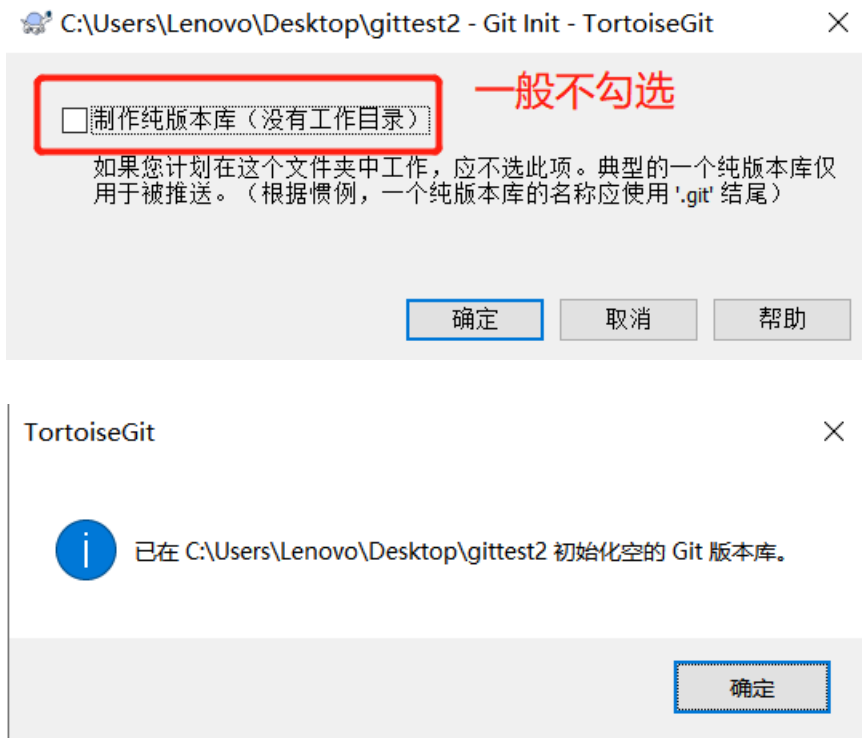
名称	修改日期	类型
.git	2019/12/9 16:17	文件夹

注意：.git 目录中存放的是本地库相关的子目录和文件，不要删除，也不要胡乱修改

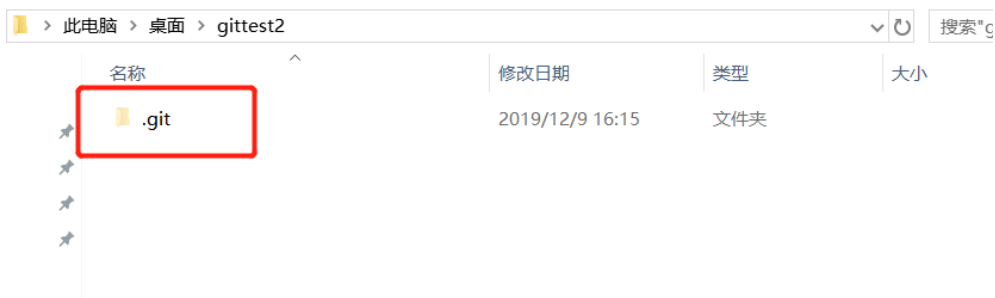
2. 使用 TortoiseGit

使用 TortoiseGit 时只需要在目录中点击右键菜单选择“在这里创建版本库”





版本库创建成功，会在此目录下创建一个.git 的隐藏目录，如下所示：



在 windows 中如何显示隐藏目录隐藏目录请自行百度 o(∩_∩)o

概念：

版本库：“.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。

(二) 设置签名

■ 形式

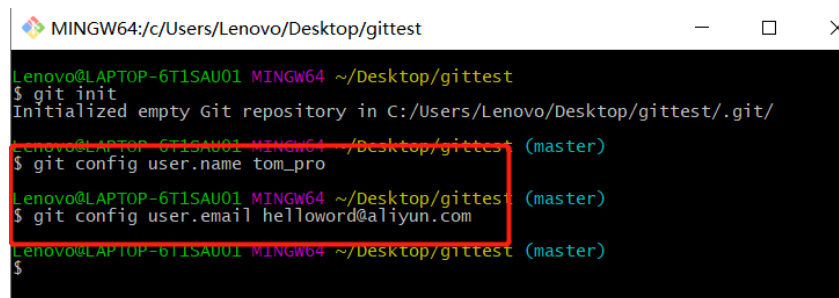
用户名：tom

Email 地址：helloword@aliyun.com

- 作用：区分不同开发人员的身份
- 辨析：这里设置的签名和登录远程库(代码托管中心)的账号、密码没有任何关系
- 命令

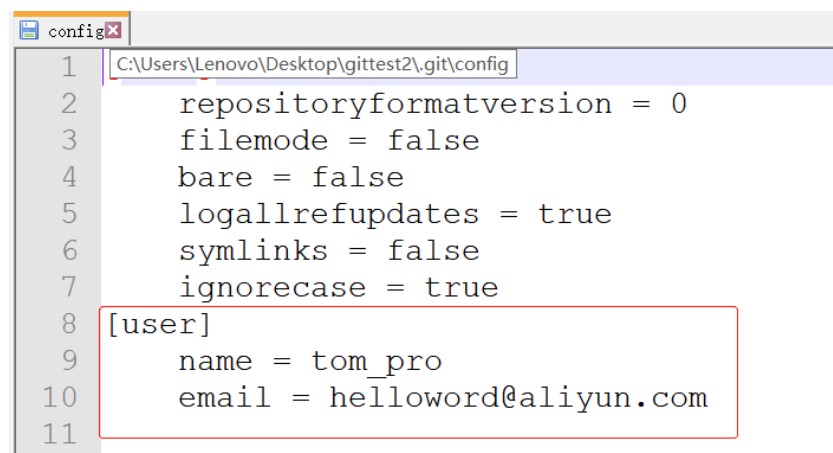
◆ 项目级别/仓库级别：仅在当前本地库范围内有效

- `git config user.name tom_pro`
- `git config user.email helloword@aliyun.com`
- 信息保存位置：.git/config 文件



```

MINGW64/c:/Users/Lenovo/Desktop/gittest
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest
$ git init
Initialized empty Git repository in C:/Users/Lenovo/Desktop/gittest/.git/
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ git config user.name tom_pro
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ git config user.email helloword@aliyun.com
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$
  
```



```

1 C:\Users\Lenovo\Desktop\gittest2\.git\config
2 repositoryformatversion = 0
3 filemode = false
4 bare = false
5 logallrefupdates = true
6 symlinks = false
7 ignorecase = true
8 [user]
9     name = tom_pro
10    email = helloword@aliyun.com
11
  
```

◆ 系统用户级别：登录当前操作系统的用户范围

- `git config --global user.name tom_global`
- `git config --global user.email tomabc@aliyun.com`

信息保存位置：保存在当前仓库目录下的 .git/config 文件中

◆ 级别优先级

- 就近原则：项目级别**优先**于系统用户级别，二者都有时采用项目级别的签名

- 如果只有系统用户级别的签名，就以系统用户级别的签名为准
- 二者都没有不允许

(三) 基本操作

1. 添加

使用 GitBash

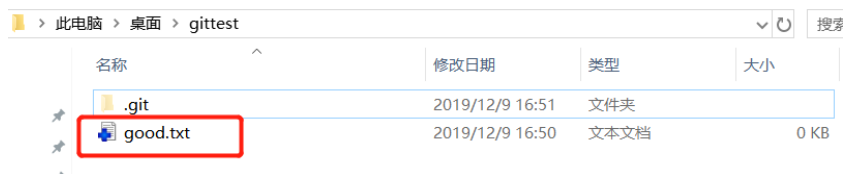
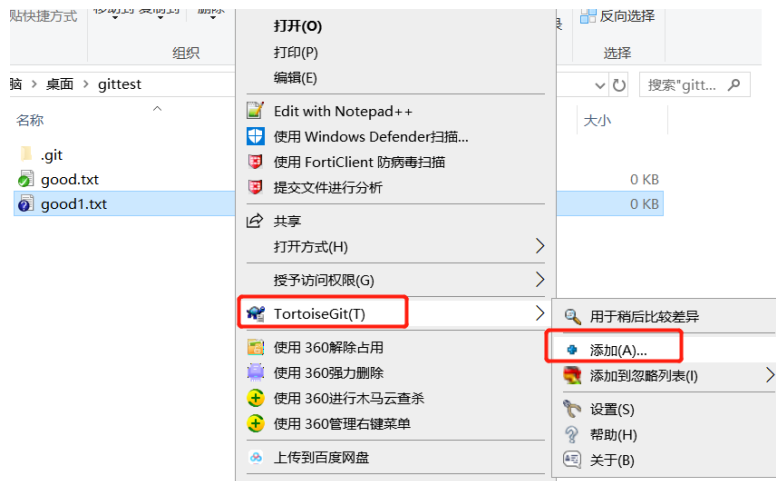
`git add [filename]`

将工作区的“新建/修改”添加到暂存区

在~/Desktop/gittest 目录下创建一个 good.txt 文件

```
MINGW64:/c:/Users/Lenovo/Desktop/gittest
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ git add good.txt
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
```

使用 TortoiseGit



名称	修改日期	类型	大小
.git	2019/12/9 16:51	文件夹	
good.txt	2019/12/9 16:50	文本文档	0 KB

2. 提交

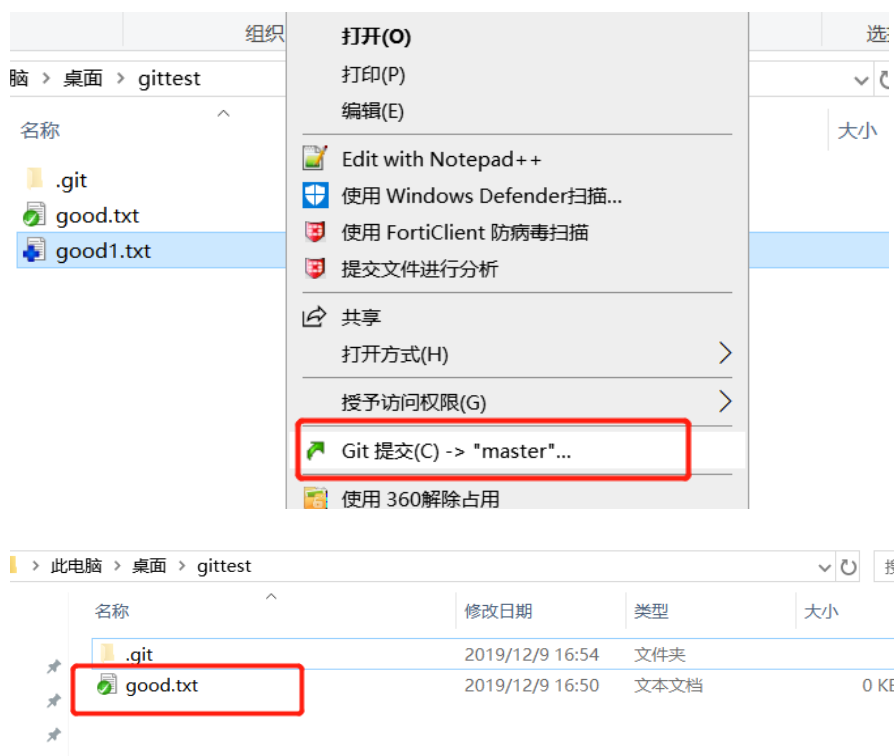
使用 GitBash

`git commit -m "commit message" [file name]`

将暂存区的内容提交到本地库（暂存区内容修改后进行提交）

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git commit good.txt
[master (root-commit) 45265b6] My first commit good.txt
1 file changed, 3 insertions(+)
create mode 100644 good.txt
```

使用 TortoiseGit



3. 查看历史记录

使用 GitBash

`git log`

```
$ git log
commit 7a6bbe572df25a239cb119d9503a61fd041e74c2 (HEAD -> master)
Author: tom_project <tomabc@aliyun.com>
Date:   Mon Feb 25 11:16:23 2019 +0800

    My second commit, insert ddddd

commit 45265b6a76d743877bf27216dc7b904a9fe49506
Author: tom_project <tomabc@aliyun.com>
Date:   Mon Feb 25 11:11:38 2019 +0800

    My first commit good.txt
```

git log --pretty=oneline 美化 log 日志

```
$ git log --pretty=oneline
7a6bbe572df25a239cb119d9503a61fd041e74c2 (HEAD -> master) My second commit, insert ddddd
45265b6a76d743877bf27216dc7b904a9fe49506 My first commit good.txt
```

git log --oneline 美化 log 日志

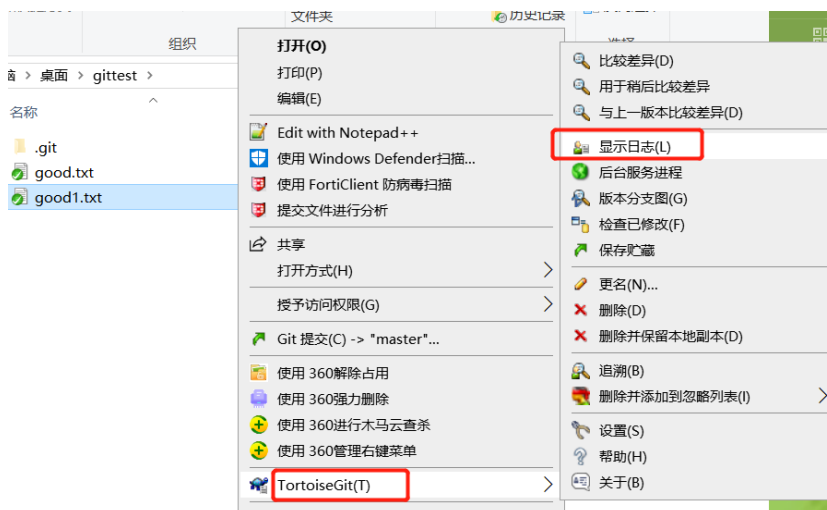
```
$ git log --oneline
7a6bbe5 (HEAD -> master) My second commit, insert ddddd
45265b6 My first commit good.txt
```

git relog (推荐)

```
$ git relog
45265b6 (HEAD -> master) HEAD@{0}: reset: moving to 45265b6
7a6bbe5 HEAD@{1}: commit: My second commit, insert ddddd
45265b6 (HEAD -> master) HEAD@{2}: commit (initial): My first commit good.txt
```

HEAD@{移动到当前版本需要多少步}

使用 TortoiseGit



4. 状态查看

使用 GitBash

git status

查看工作区，暂存区状态

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git status
On branch master

No commits yet

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

    good.txt

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

文件名红色表明工作目录中的文件尚未被追踪管理

■ 前进后退

◆ 本质[了解]

其实内部维护了一个 HEAD 指针，做这么一个 reset 操作的时候，其实相当于易懂了 HEAD 指针（表明的是当前的状态）

◆ 基于索引值操作[推荐]

git reset --hard [局部索引值/版本号前缀]

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git reflog
7a6bbe5 (HEAD -> master) HEAD@{0}: commit: My second commit, insert ddddd
45265b6 HEAD@{1}: commit (initial): My first commit good.txt

IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git reset --hard 45265b6
HEAD is now at 45265b6 My first commit good.txt
```

◆ 使用^符号：只能后退

git reset --hard HEAD^ 一个表示后退一步，n 个表示后退 n 步

◆ 使用~符号：只能后退

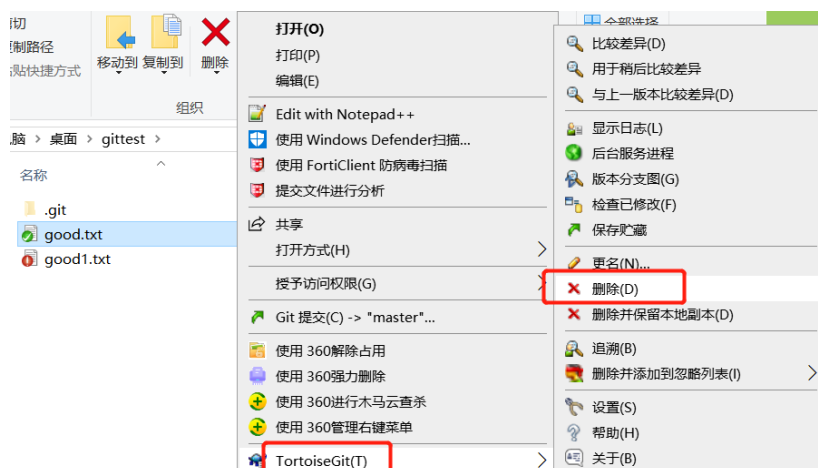
git reset --hard HEAD~n

注：表示后退 n 步

5. 删除文件

使用 TortoiseGit

需要删除无用的文件时可以使用 git 提供的删除功能直接将文件从版本库中删除。



注意:不要仅仅从工作目录删除,删除之后也需要 git add 然后 git commit 提交到本地仓库管理

6. 删除文件并找回

使用 GitBash

- 前提: 删除前, 文件存在时的状态提交到了本地库
- 操作: git reset --hard 指针位置
 - ◆ 删除操作已经提交到本地库: 指针位置指向历史记录

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git reflog
fcd29ec (HEAD -> master) HEAD@{0}: commit: delete good.txt
45265b6 HEAD@{1}: reset: moving to HEAD^
7a6bbe5 HEAD@{2}: reset: moving to 7a6bbe5
45265b6 HEAD@{3}: reset: moving to 45265b6
7a6bbe5 HEAD@{4}: commit: My second commit, insert dddd
45265b6 HEAD@{5}: commit (initial): My first commit good.txt

IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git reset --hard 7a6bbe5
HEAD is now at 7a6bbe5 My second commit, insert dddd

IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
```

7. 比较文件差异

使用 GitBash

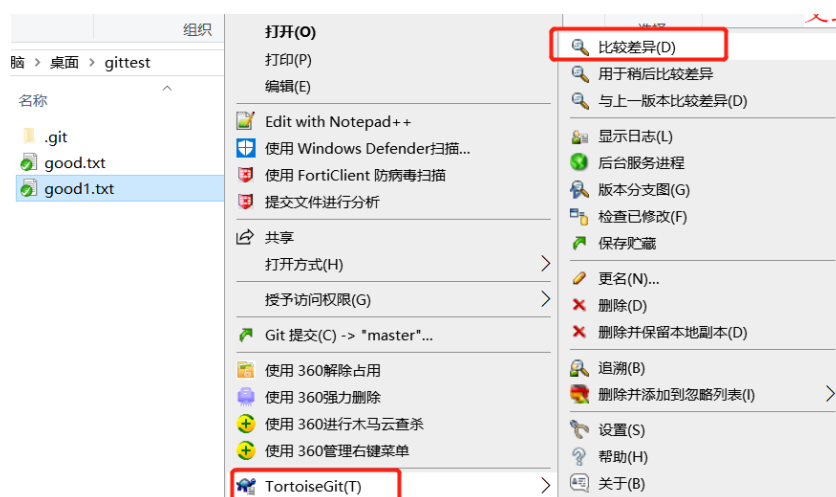
- git diff [文件名]: 将工作区中的文件和暂存区进行比较

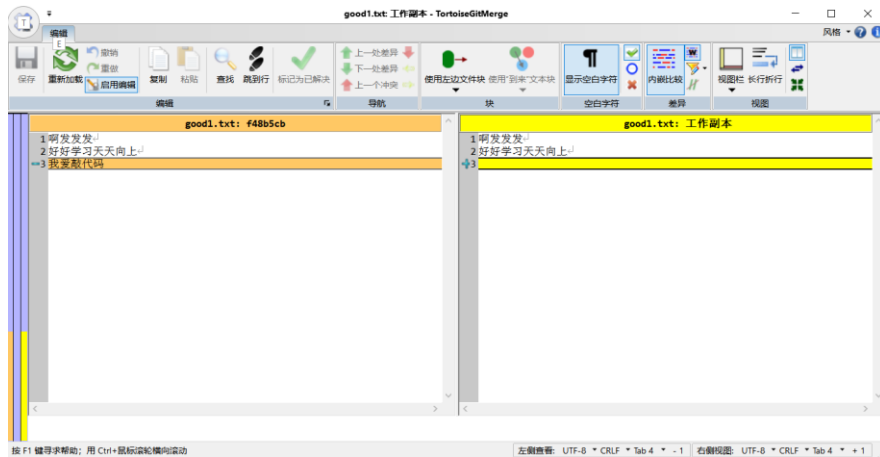
```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git diff good.txt
diff --git a/good.txt b/good.txt
index 8fd5a9e..9207182 100644
--- a/good.txt
+++ b/good.txt
@@ -2,4 +2,5 @@ aaaaaaaaaa
 bbbbbbbbbb
 ccccccccc
 dddddddd
-eeeeeeee
\ No newline at end of file
+eeeeeeee
+ffffff
\ No newline at end of file
```

git diff 本地仓库中的历史版本 文件名：将工作区中的文件和版本库进行比较(主要)

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git diff 7a6bbe5 good.txt
diff --git a/good.txt b/good.txt
index 5e7421b..9207182 100644
--- a/good.txt
+++ b/good.txt
@@ -1,4 +1,6 @@
 aaaaaaaaaa
 bbbbbbbbbb
 ccccccccc
-dddddddd
\ No newline at end of file
+dddddddd
+eeeeeeee
+ffffff
\ No newline at end of file
```

使用 TortoiseGit

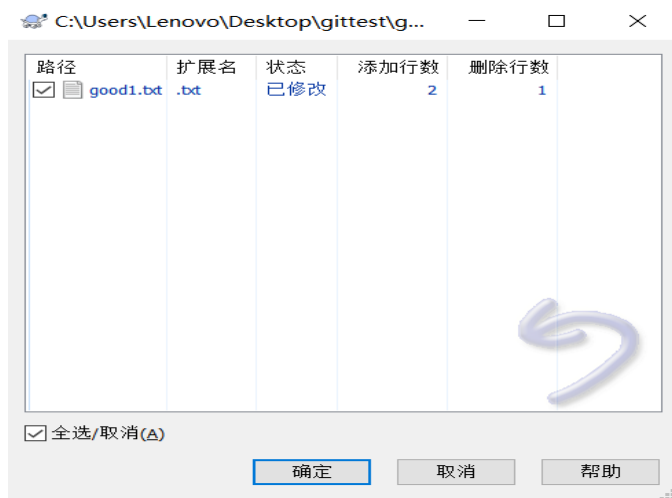
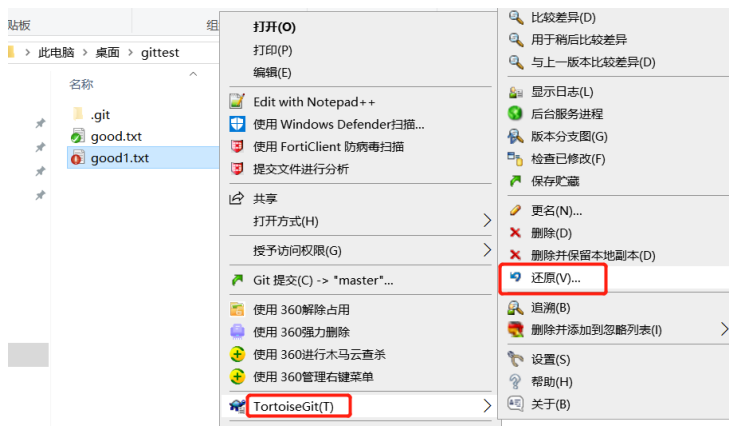


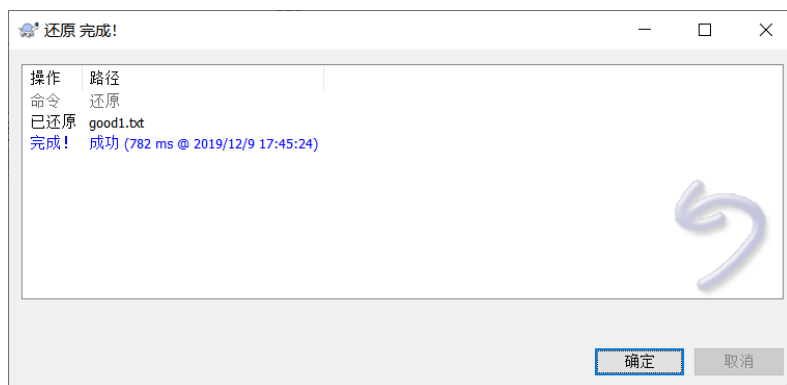


8. 还原修改

使用 TortoiseGit

当文件修改后不想把修改的内容提交，还想还原到未修改之前的状态。此时可以使用“还原”功能





注意：此操作会撤销所有未提交的修改，所以当做还原操作是需要慎重慎重！！

(四) 工作区和暂存区

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。

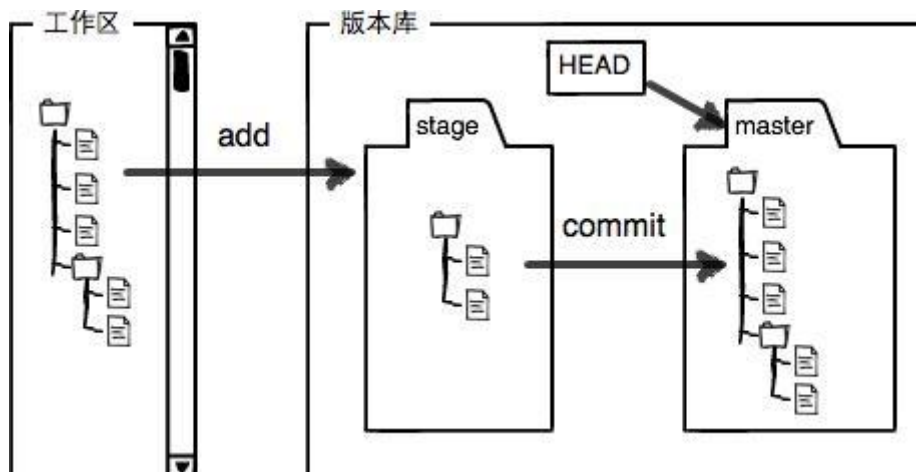
什么是工作区 (Working Directory) ?

工作区就是你在电脑里能看到的目录，比如我的 repository 文件夹就是一个工作区。

有的同学可能会说 repository 不是版本库吗怎么是工作区了？其实 repository 目录是工作区，在这个目录中的 “.git” 隐藏文件夹才是版本库。这回概念清晰了吧。

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 的概念我们稍后再讲。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。

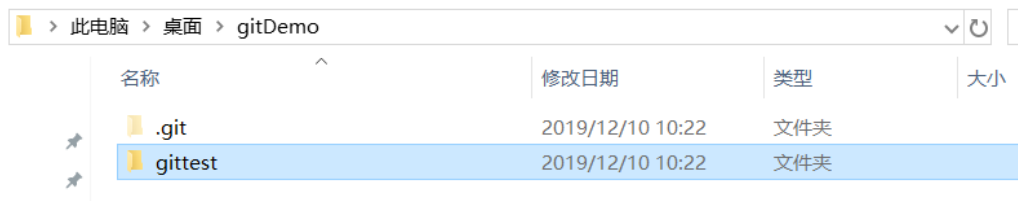
因为我们创建 Git 版本库时，Git 自动为我们创建了唯一一个 `master` 分支，所以，现在，`git commit` 就是往 `master` 分支上提交更改。

你可以简单理解为，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。

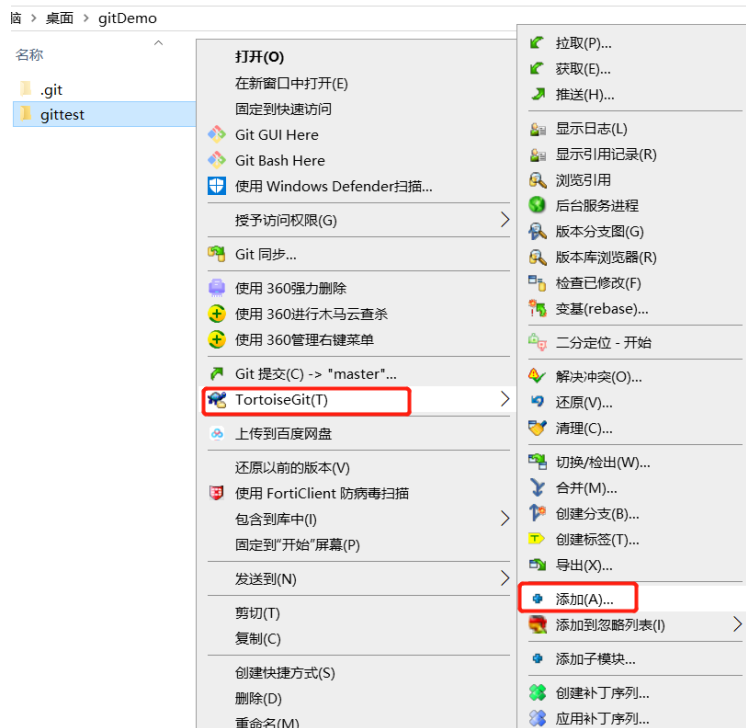
(五) 案例：将 java 工程提交到版本库

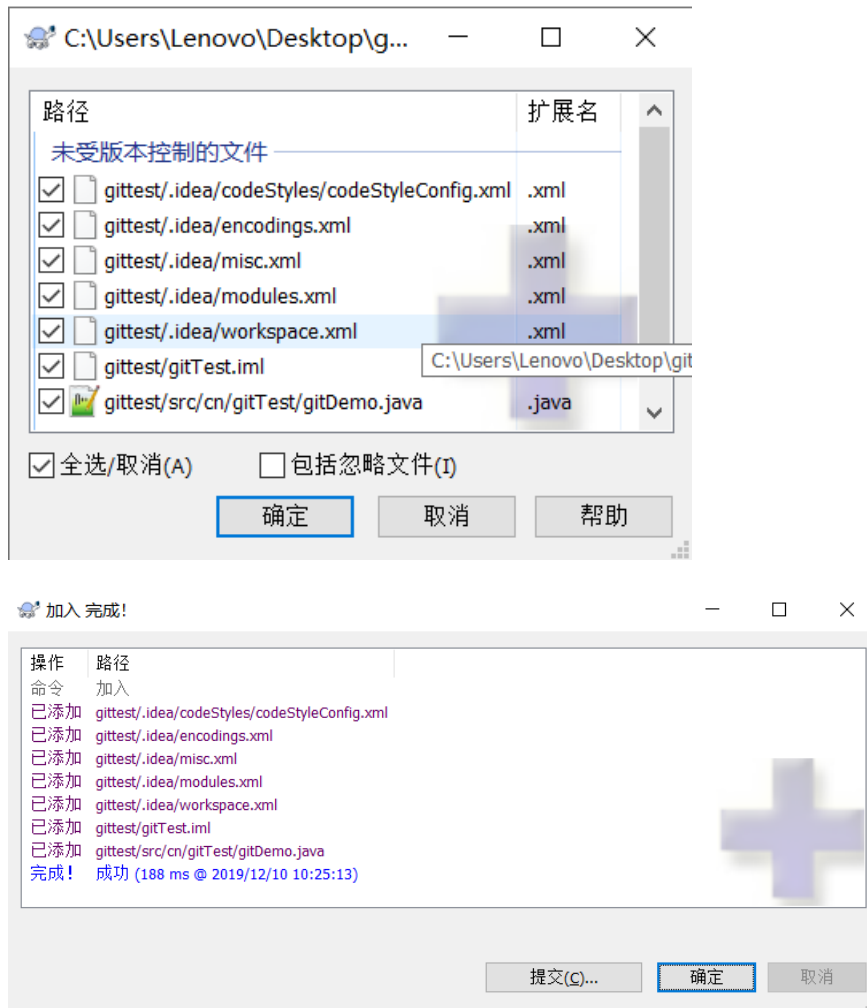
使用 TortoiseGit

第一步：将参考资料中的 java 工程 `gittest` 复制到工作目录中



第二步：将工程添加到暂存区。



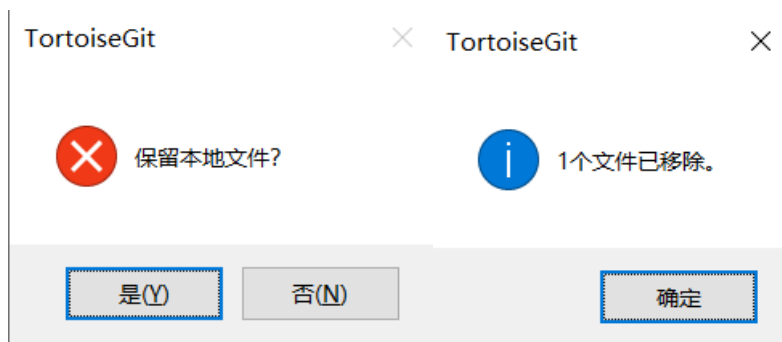
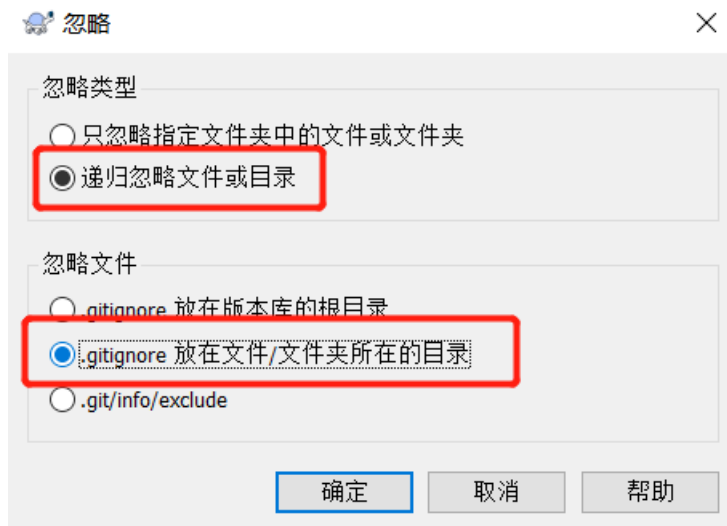


点击确定完成暂存区添加。

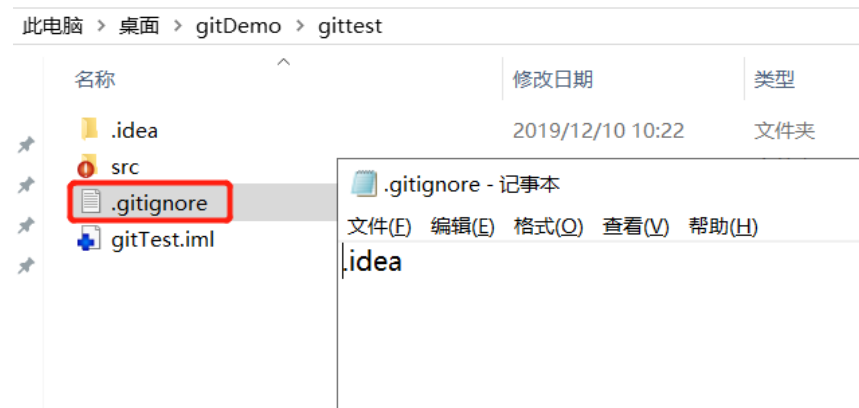
三、忽略文件或文件夹

在此工程中，并不是所有文件都需要保存到版本库中的例如“.idea”目录及目录下的文件就可以忽略。好在 Git 考虑到了大家的感受，这个问题解决起来也很简单，在 Git 工作区的根目录下创建一个特殊的.gitignore 文件，然后把要忽略的文件名填进去，Git 就会自动忽略这些文件。

如果使用 TortoiseGit 的话可以使用菜单项直接进行忽略。



选择保留本地文件。完成后在此文件夹内会多出一个.gitignore 文件，这个文件就是文件忽略文件，当然也可以手工编辑。其中的内容就是把.idea 目录忽略掉。



四、提交代码

将代码添加到 master 分支上，其中.gitignore 文件也需要添加到暂存区，然后提交到版本库。

(六) 忽略文件语法规则

空行或是以 # 开头的行即注释行将被忽略。

可以在前面添加正斜杠 / 来避免递归,下面的例子中可以很明白的看出来与下一条的区别。

可以在后面添加正斜杠 / 来忽略文件夹，例如 build/ 即忽略 build 文件夹。

可以使用 ! 来否定忽略，即比如在前面用了 *.apk ，然后使用 !a.apk ，则这个 a.apk 不会被忽略。

* 用来匹配零个或多个字符，如 *.oa] 忽略所有以".o"或".a"结尾， *~ 忽略所有以 ~ 结尾的文件（这种文件通常被许多编辑器标记为临时文件）； [] 用来匹配括号内的任一字符，如 [abc] ，也可以在括号内加连接符，如 [0-9] 匹配 0 至 9 的数； ? 用来匹配单个字符。

看了这么多，还是应该来个栗子：

忽略 .a 文件

*.a

但否定忽略 lib.a, 尽管已经在前面忽略了 .a 文件

!lib.a

仅在当前目录下忽略 TODO 文件， 但不包括子目录下的 subdir/TODO

/TODO

忽略 build/ 文件夹下的所有文件

build/

忽略 doc/notes.txt, 不包括 doc/server/arch.txt

```
doc/*.txt
```

```
# 忽略所有的 .pdf 文件 在 doc/ directory 下的
```

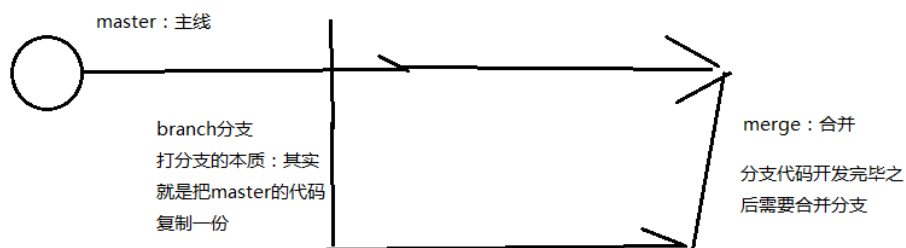
```
doc/**/*.pdf
```

七、Git 分支管理[了解]

往往应用于大型项目，而且分支由项目经理管理

(一) 什么是分支

在版本控制过程中，使用多条线同时推进多个任务



打分支：产品经理或者用户提出了新的不确定（需求本身什么样不确定、这个需求最终上线否不确定）的需求，如果我们还在master上面开发，势必会影响已经稳健的项目代码了（污染原来的代码）

打分支的另外一个场景：master主分支发现了bug不容易解决，打分支专门fix bug，等bug fix完毕之后再merge到主分支

(二) 分支的好处

- 各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响，失败的分支删除重新开始即可

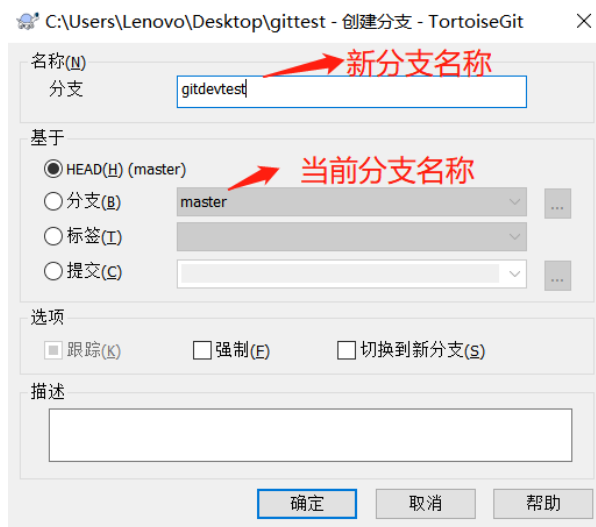
(三) 分支操作

1. 创建分支

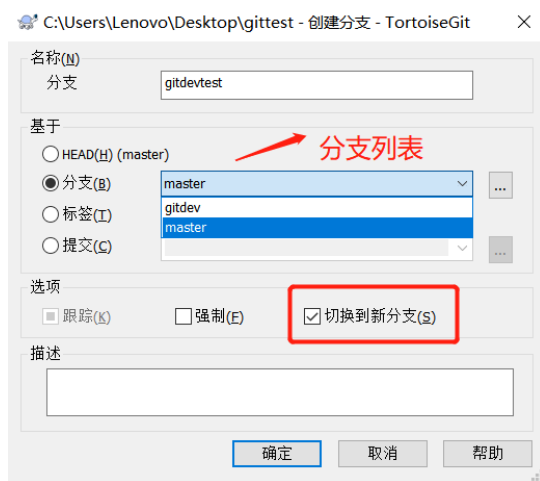
使用 **GitBash**

```
git branch 分支名
```

使用 TortoiseGit



如果想创建完毕后直接切换到新分支可以勾选“切换到新分支”选项或者从菜单中选择“切换/检出”来切换分支：



2. 查看分支

使用 GitBash

```
git branch -v
```

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git branch -v
* master 9d0c526 commit
```

3. 切换分支（checkout 检出）

使用 GitBash

```
git checkout 【分支名】
```

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (master)
$ git checkout fixbug
Switched to branch 'fixbug'
```


ll命令 可查看分支中的文件
 编辑文件我们超前借用linux中的文本编辑器vim
 vim good.txt打开文件
 点击键盘上的i (insert) 进入编辑模式
 编辑完成后需要保存
 按esc退出编辑模式
 按shift+:, 输入wq

 cat 文件名:用来查
 看文件,不能编辑

4. 合并分支

使用 GitBash

第一步: 切换到接受修改的分支

git checkout 【被合并分支名】

第二部: 执行 merge (合并) 命令

git merge 【有新内容分支名】

```
MINGW64~/c/Users/Lenovo/Desktop/gittest
total 2
-rw-r--r-- 1 Lenovo 197121 2 12月 9 18:09 a.txt
-rw-r--r-- 1 Lenovo 197121 0 12月 9 16:50 good.txt
-rw-r--r-- 1 Lenovo 197121 40 12月 9 17:47 good1.txt
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (gitdev)
$ git checkout master
Switched to branch 'master'
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ ll
total 1
-rw-r--r-- 1 Lenovo 197121 0 12月 9 16:50 good.txt
-rw-r--r-- 1 Lenovo 197121 55 12月 9 18:10 good1.txt
Lenovo@LAPTOP-6T1SAU01 MINGW64 ~/Desktop/gittest (master)
$ git merge gitdev
Updating 51865cb..f3634f9
Fast-forward
 a.txt      | 1 +
 good1.txt | 1 -
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 a.txt
```

5. 解决冲突

使用 GitBash

- 冲突的表现
- 冲突的解决

第一步：编辑文件，删除特殊符号

第二步：把文件修改到满意的程度，保存退出

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (fixbug|MERGING)
$ cat good.txt
aaaaaaaaaa
bbbbbbbbbb
cccccccccc
dddddddddd
eeeeeeeeee
ffffffffff
gggggggggg
<<<<<<< HEAD
jjjjjjjj-by fixbug
=====
hhhhhhhh-by master
>>>>>>> master
```

不要留有特殊字符<<<<<<<=>>>>>>>

第三步：git add 文件名

第四步：git commit -m '日志信息'，注意此时，commit 一定不能带具体的文件名,否则会出现错误

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/gitDemo (fixbug|MERGING)
$ git commit -m "process merge conflict" good.txt
fatal: cannot do a partial commit during a merge.
```

八、GitHub[应用]

(一) 添加远程仓库

现在我们已经在本地创建了一个 Git 仓库，又想让其他人来协作开发，此时就可以把本地仓库同步到远程仓库，同时还增加了本地仓库的一个备份。

常用的远程仓库就是 github: <https://github.com/>，接下来我们演示如何将本地代码同步到 github。

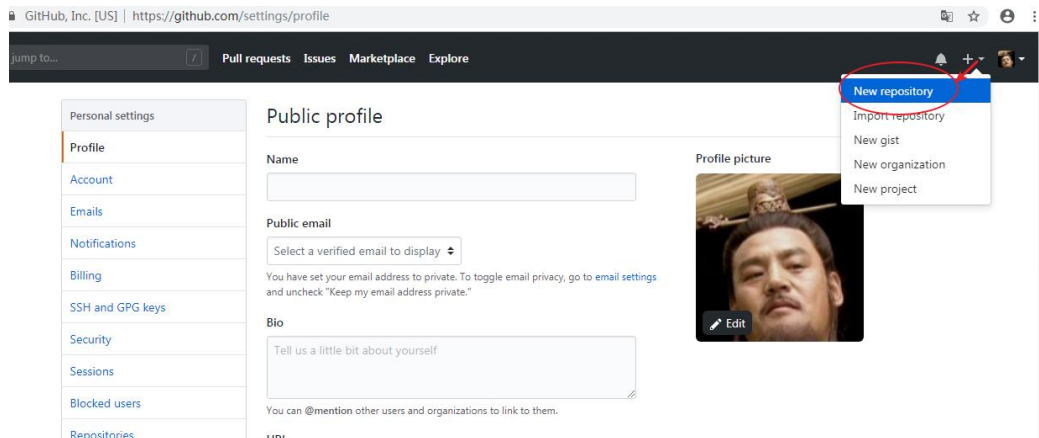
(二) 账号信息

GitHub 首页就是注册页面: <https://github.com/>

首先你得在 github 上创建一个账号

	<p>Email 地址: githubyuebuqun@aliyun.com</p> <p>GitHub 账号: githubyuebuqun</p>
	<p>Email 地址: githublinghuchong@aliyun.com</p> <p>GitHub 账号: githublinghuchong</p>
	<p>Email 地址: githubdongfangbubai@aliyun.com</p> <p>GitHub 账号: githubdongfangbubai</p>

(三) 创建 GitHub 远程库



Create a new repository

A repository contains all project files, including the revision history.

Owner: githubbyuebuqun

Repository name *: huashan1105 ✓

Great repository names are short and memorable. Need inspiration? How about **expert-adventure**?

Description (optional):

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

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

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

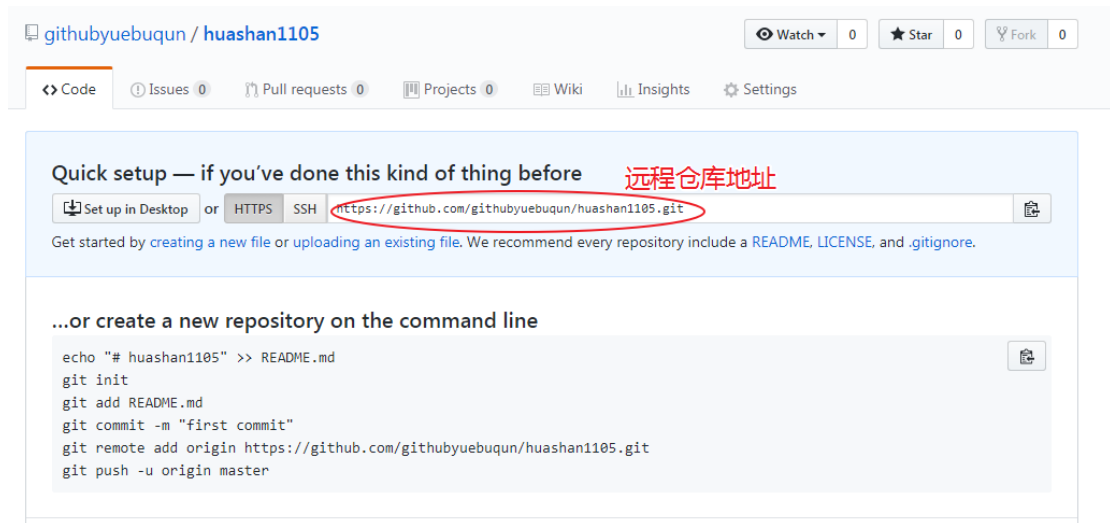
Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

点击“create repository”按钮仓库就创建成功了。

Github 支持两种同步方式“https”和“ssh”。如果使用 https 很简单基本不需要配置就可以使用，但是每次提交代码和下载代码时都需要输入用户名和密码。如果使用 ssh 方式就需要客户端先生成一个密钥对，即一个公钥一个私钥。然后还需要把公钥放到 github 的服务器上。这两种方式在实际开发中都应用，所以我们都掌握。

(四) 创建远程仓库地址别名



git remote -v 查看当前所有远程地址别名

git remote add [别名] [远程地址]

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master)
$ git remote add huashanRemote https://github.com/githubyuebuqun/huashan1105.git

IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master)
$ git remote -v
huashanRemote https://github.com/githubyuebuqun/huashan1105.git (fetch)
huashanRemote https://github.com/githubyuebuqun/huashan1105.git (push)
```

(五) 同步到远程仓库

https 方式

git push [别名] [分支名]

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master)
$ git push huashanRemote master
Username for 'https://github.com': githubyuebuqun
Counting objects: 3, done.
Writing objects: 100% (3/3), 260 bytes | 260.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/githubyuebuqun/huashan1105.git
 * [new branch] master -> master
```

SSH 方式

git remote rm origin

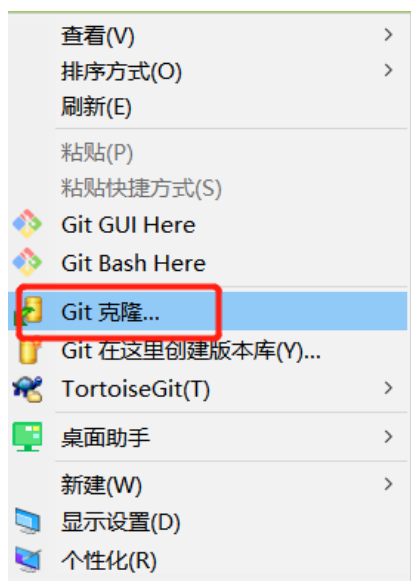
git remote add origin [ssh 仓库地址]

git push -u origin [分支]

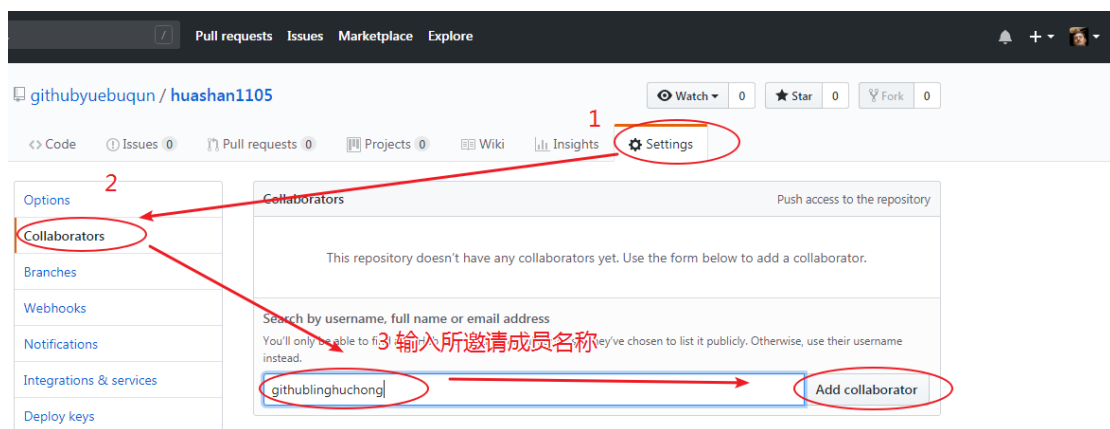
(六) 从远程仓库克隆

- 命令: `git clone [仓库远程地址]`
- 效果
 - 完整的把远程库下载到本地
 - 创建 origin 远程地址别名
 - 初始化本地库

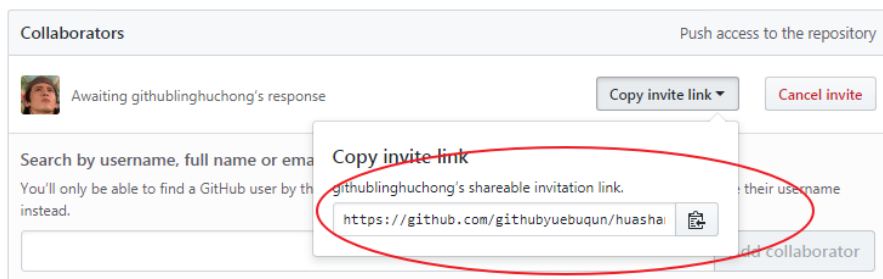
使用 TortoiseGit:



(七) 团队成员邀请



复制邀请连接



“岳不群”其他方式把邀请连接发送给“令狐冲”，“令狐冲”登录自己的 GitHub 账号，访问邀请连接，接受邀请



(八) 从远程仓库取代码

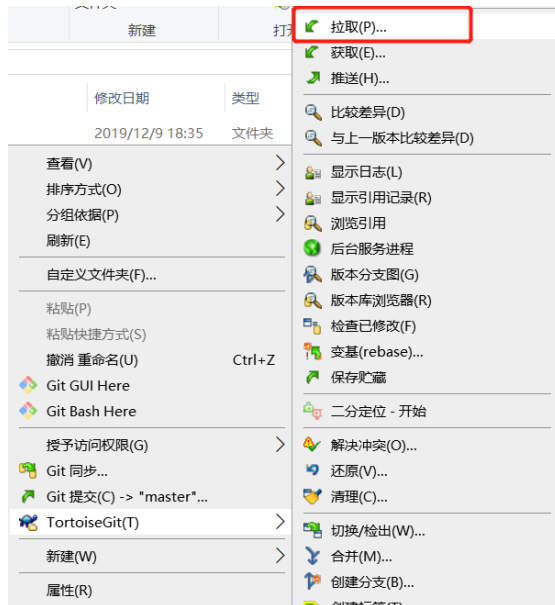
- pull = fetch（获取的意思） + merge
- git fetch [远程库地址别名] [远程分支名]
 - 如果要查看，需要切换仓库 git checkout origin/master,然后 cat 文件

例如：

```
git fetch original master
git checkout original/master
cat ....txt
--git checkout master
git merge original/master
```

- git merge [远程库地址别名/远程分支名]
- git pull [远程库地址别名] [远程分支名]

或



(九) 解决冲突

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master)
$ git push huashanRemote master
Username for 'https://github.com': githubyuebuqun
To https://github.com/githubyuebuqun/huashan1105.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/githubyuebuqun/huashan1105.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.
```

首先需要 pull 最新的代码


```
IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master)
$ git pull huashanRemote master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/githubyuebuqun/huashan1105
  * branch                master      -> FETCH_HEAD
  2a40647..f7a04d2 master      -> huashanRemote/master
Auto-merging huashanjianfa.txt
CONFLICT (content): Merge conflict in huashanjianfa.txt
Automatic merge failed; fix conflicts and then commit the result.

IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master|MERGING)
$ ll
total 1
-rw-r--r-- 1 IBM 197121 225 二月 25 15:48 huashanjianfa.txt

IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105 (master|MERGING)
$ cat huashanjianfa.txt
华山剑法，天下第一！
我是令狐冲，大师兄，剑法第一！
<<<<<<< HEAD
我是岳不群，我想练习葵花宝典
=====
我会独孤九剑！牛
>>>>>>> f7a04d2aa8895f7d03a1dc39d81910b3b35e3c18
```

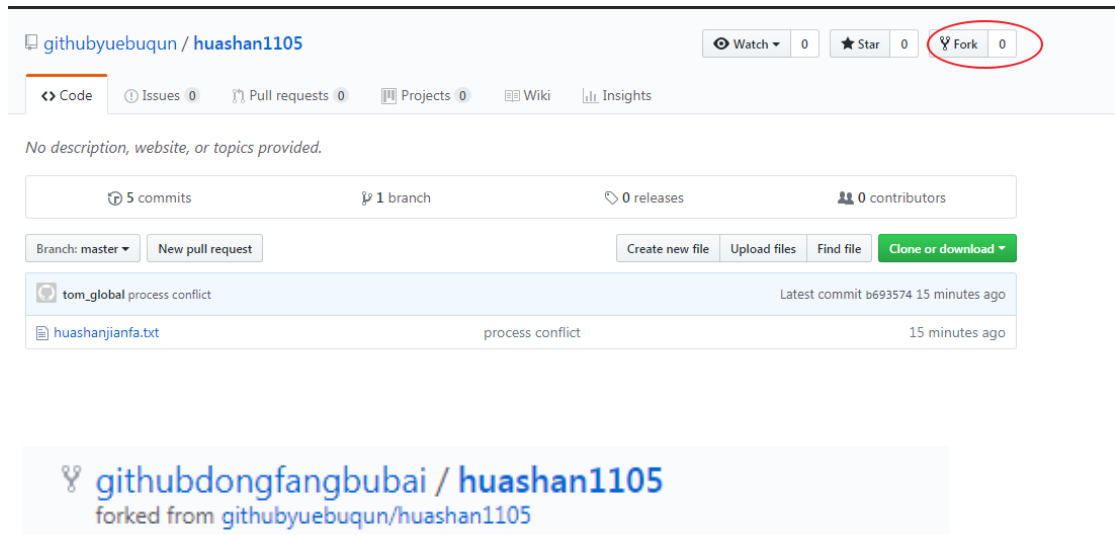
然后处理冲突，和分支合并处理冲突一样，注意 commit 时不要带文件名，commit 后 push 到远程

■ 要点：

- 如果不是基于 GitHub 远程库的最新版所做的修改，不能推送，必须先拉取
- 拉取下来如果进入冲突状态，模仿分支冲突解决即可

(十) 跨团队协作

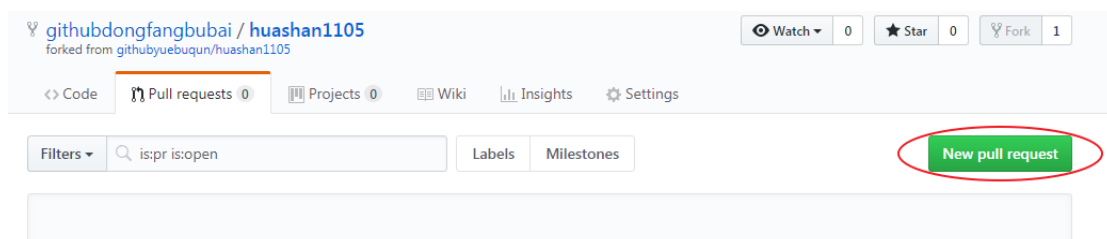
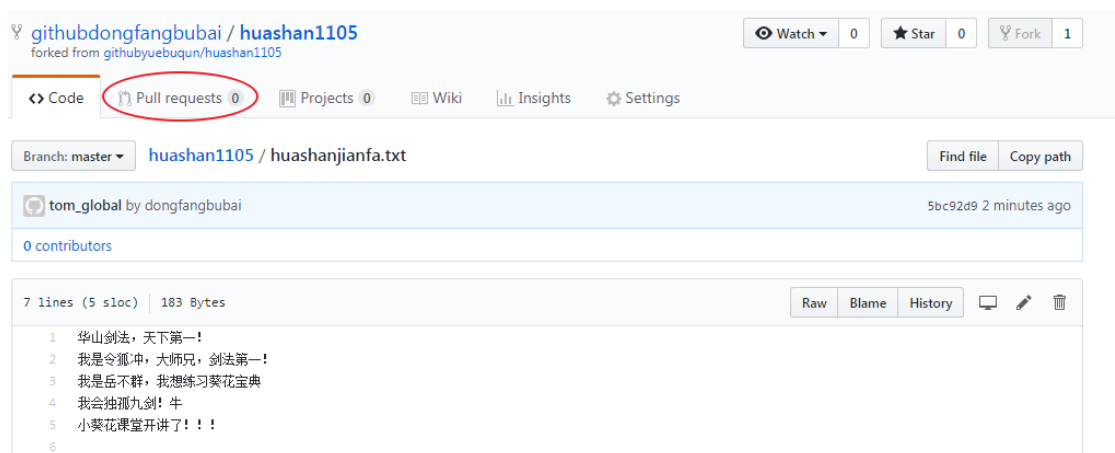
- 把项目的地址发送给团队外部的人（线下发送）
- 团队外部人员登录 github，然后访问给出的地址
- 团队外人员把 fork 后的远程仓库 clone 到本地进行 fork



- 团队外人员本地编辑提交，push 到远程

```
IBM@IBM-PC MINGW64 ~/Desktop/repos/huashan1105_dfbb/huashan1105 (master)
$ git push origin master
Username for 'https://github.com': githubdongfangbubai
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 313.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/githubdongfangbubai/huashan1105.git
b693574..5bc92d9 master -> master
```

- pull request



CodeIssues 0Pull requests 0Projects 0WikiInsights

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base repository: githubyuebuqun/huashan1105base: master

head repository: githubdongfangbubai/huasha...compare: master

✓ Able to merge. These branches can be automatically merged.

Create pull requestDiscuss and review the changes in this comparison with others.

1 commit1 file changed0 commit comments1 contributor

Commits on Feb 25, 2019

tom_globalby dongfangbubai5bc92d9

✓ Able to merge. These branches can be automatically merged.

老岳，你托我办的事整完了

WritePreview

AA B i “ < > @

菱花宝典天下第一，快练吧

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

✓ Allow edits from maintainers. Learn more

Create Pull Request

类似于该请求的注释

- 团队内人员，查看审核请求

团队人员登录

CodeIssues 0Pull requests 1Projects 0WikiInsightsSettings

No description, website, or topics provided.

Manage topics

5 commits1 branch0 releases0 contributors

Branch: masterNew pull request

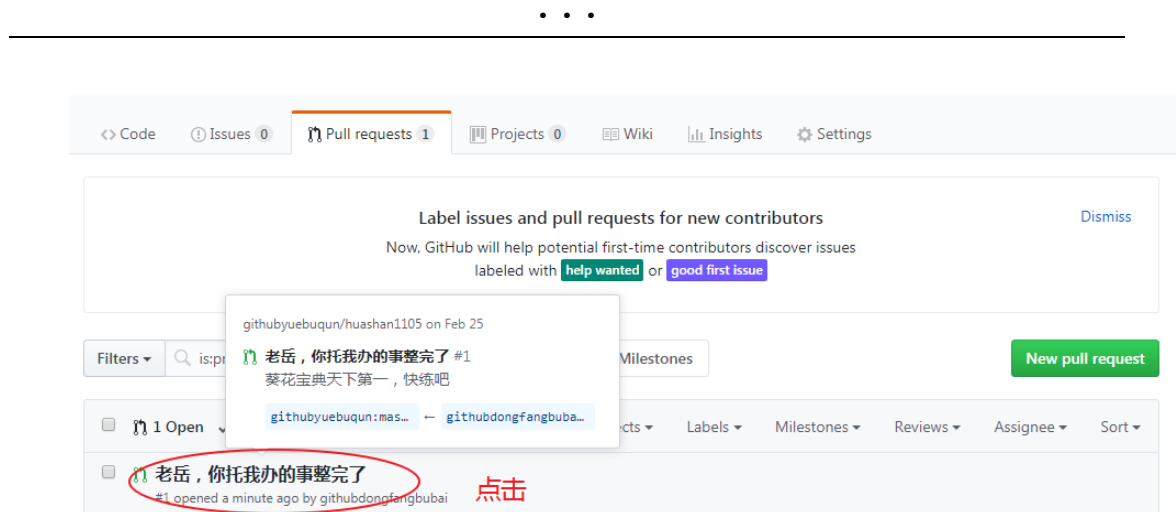
Create new fileUpload filesFind fileClone or download

tom_global process conflictLatest commit b693574 25 minutes ago

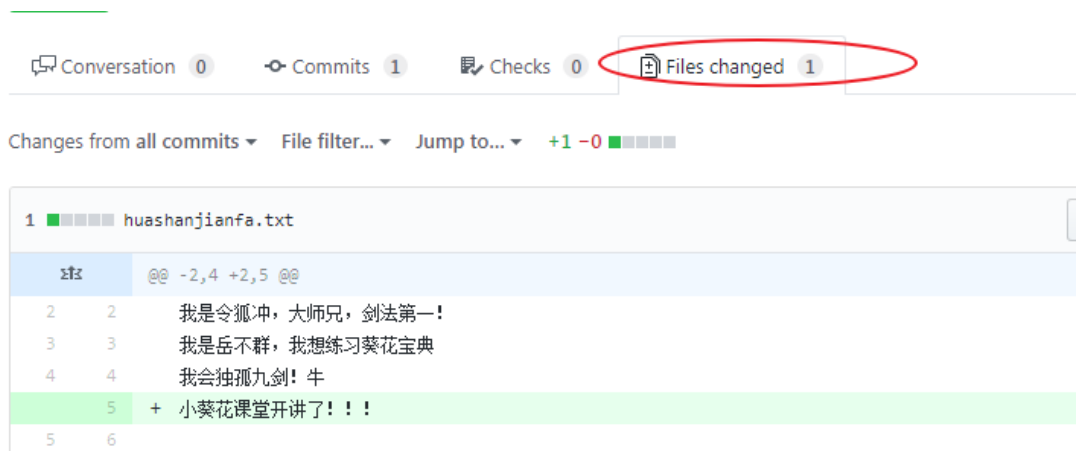
huashanjanfa.txtprocess conflict25 minutes ago

Help people interested in this repository understand your project by adding a README.

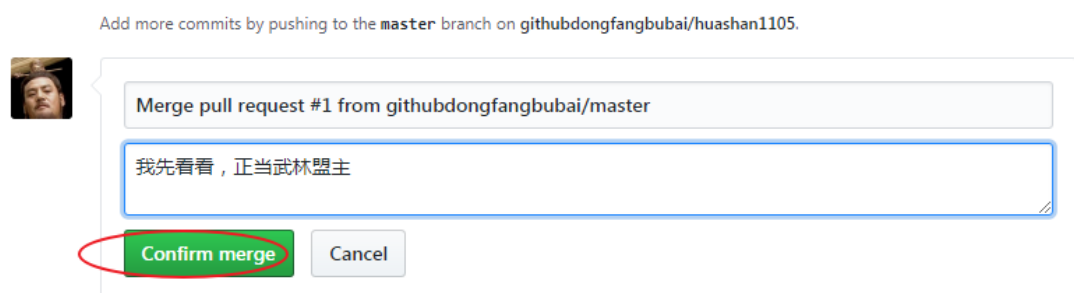
Add a README



查看修改内容



- 然后将远程库拉取到本地即可，拉取到本地结束跨团队协作的过程



(十一) SSH 操作

1. 什么是 ssh?

SSH 为 Secure Shell(安全外壳协议)的缩写,由 IETF 的网络小组(Network Working Group)所制定。SSH 是目前较可靠,专为远程登录会话和其他网络服务提供安全性的协议。利用

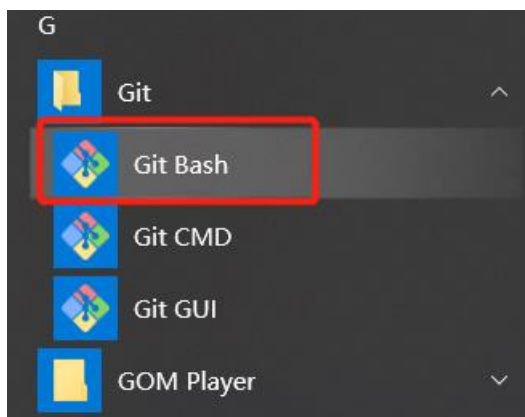
SSH 协议可以有效防止远程管理过程中的信息泄露问题。

2. 基于密匙的安全验证

使用 ssh 协议通信时，推荐使用基于密钥的验证方式。你必须为自己创建一对密匙，并把公用密匙放在需要访问的服务器上。如果你要连接到 SSH 服务器上，客户端软件就会向服务器发出请求，请求用你的密匙进行安全验证。服务器收到请求之后，先在该服务器上你的主目录下寻找你的公用密匙，然后把它和你发送过来的公用密匙进行比较。如果两个密匙一致，服务器就用公用密匙加密“质询”（challenge）并把它发送给客户端软件。客户端软件收到“质询”之后就可以用你的私人密匙解密再把它发送给服务器。

(1) ssh 密钥生成

在 windows 下我们可以使用 Git Bash.exe 来生成密钥，可以通过开始菜单或者右键菜单打开 Git Bash



git bash 执行命令,生成公钥和私钥

命令: **ssh-keygen -t rsa**

```
MINGW64: /d/temp/git/repository
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): 敲回车即可
Enter passphrase (empty for no passphrase): 一路回车
Enter same passphrase again: 使用默认配置
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa.
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:8a0c1ru1hk5J8WN7+pRouB4Jpvt4XINugNO2ZJFRG3A Administrator@PC-201311301552
The key's randomart image is:
+---[RSA 2048]---+
  .oE
  .. o
  +. .
  o o o o
  o So= + +
  o *O+.*O+.o.
  =.= ++=+o.o
  .o= .o.=
  o+..+o.o..
+-----[SHA256]-----+

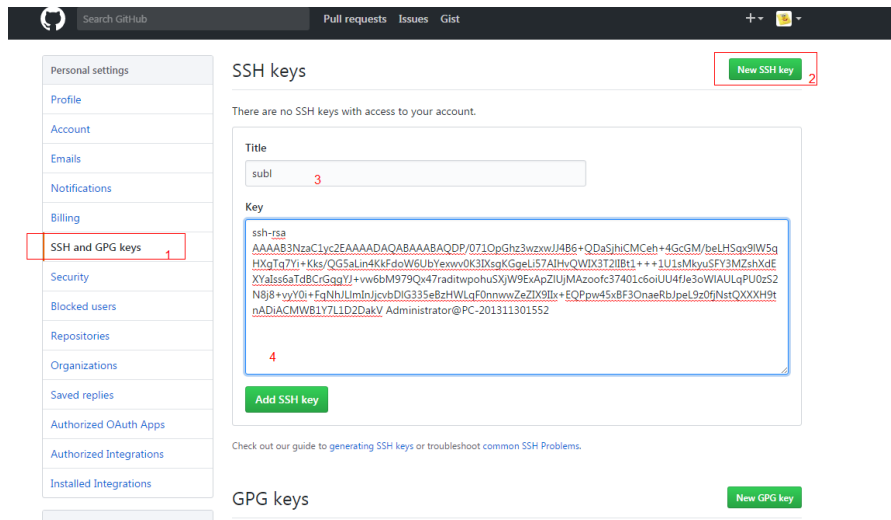
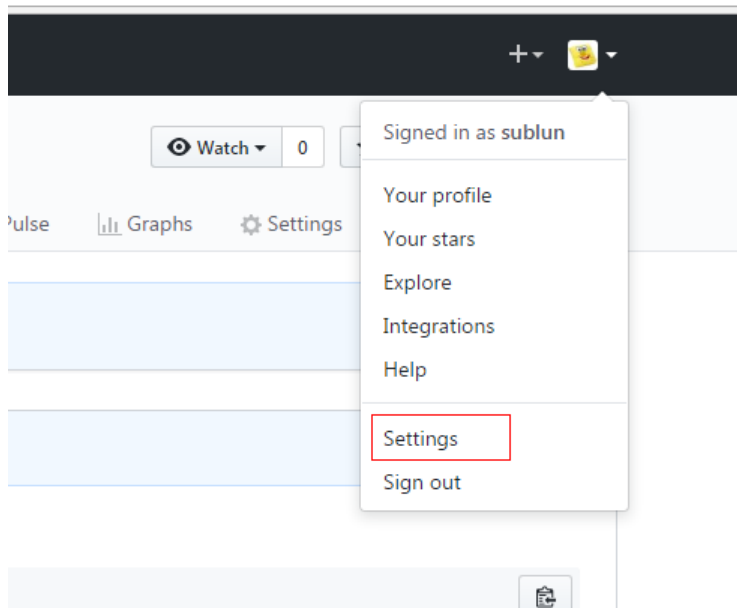
Administrator@PC-201311301552 MINGW64 /d/temp/git/repository (master)
$
```

执行命令完成后,在 window 本地用户.ssh 目录 C:\Users\用户名\.ssh 下面生成如下名称的公钥和私钥:

id_rsa	私钥	2019/12/10 10:57	文件
id_rsa.pub	公钥	2019/12/10 10:57	PUB 文件

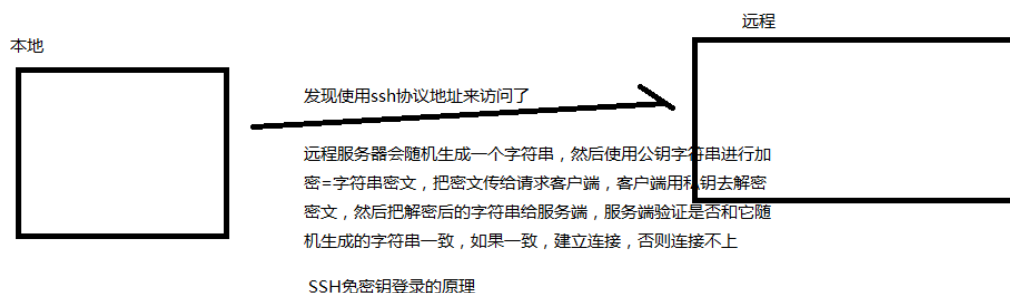
(2) ssh 密钥配置

密钥生成后需要在 github 上配置密钥本地才可以顺利访问。



在 key 部分将 id_rsa.pub 文件内容添加进去，然后点击“Add SSH key”按钮完成配置。

3. SSH 原理

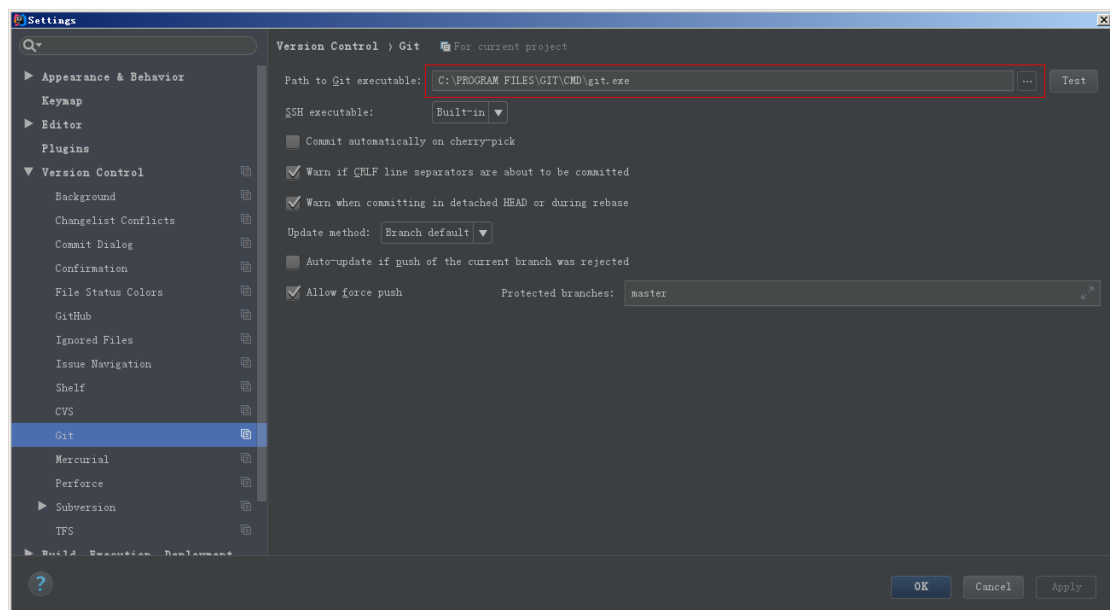


九、Idea 中操作 Git[掌握]

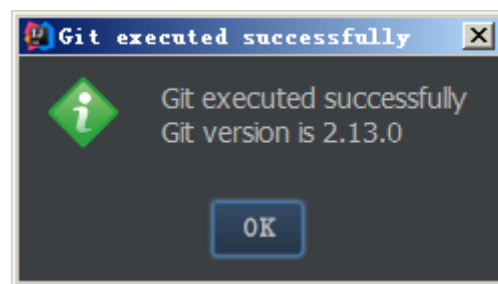
(一) 在 Idea 中配置 git

安装好 IntelliJ IDEA 后，如果 Git 安装在默认路径下，那么 idea 会自动找到 git 的位置，如果更改了 Git 的安装位置则需要手动配置下 Git 的路径。

选择 File→Settings 打开设置窗口，找到 Version Control 下的 git 选项：

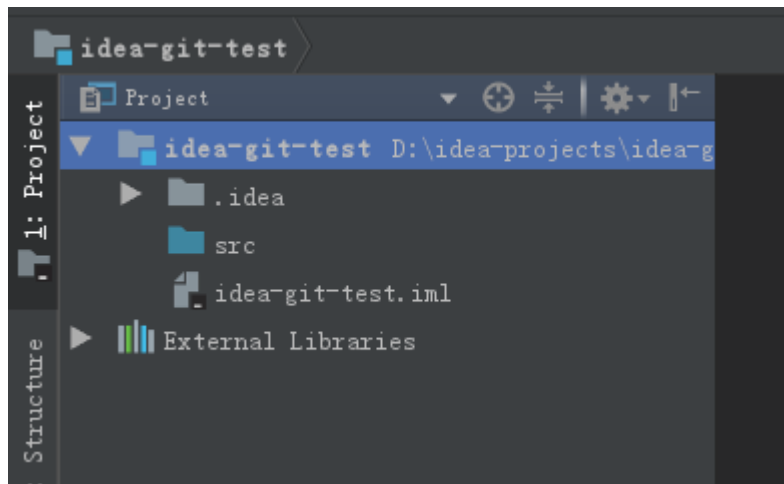


选择 git 的安装目录后可以点击“Test”按钮测试是否正确配置。



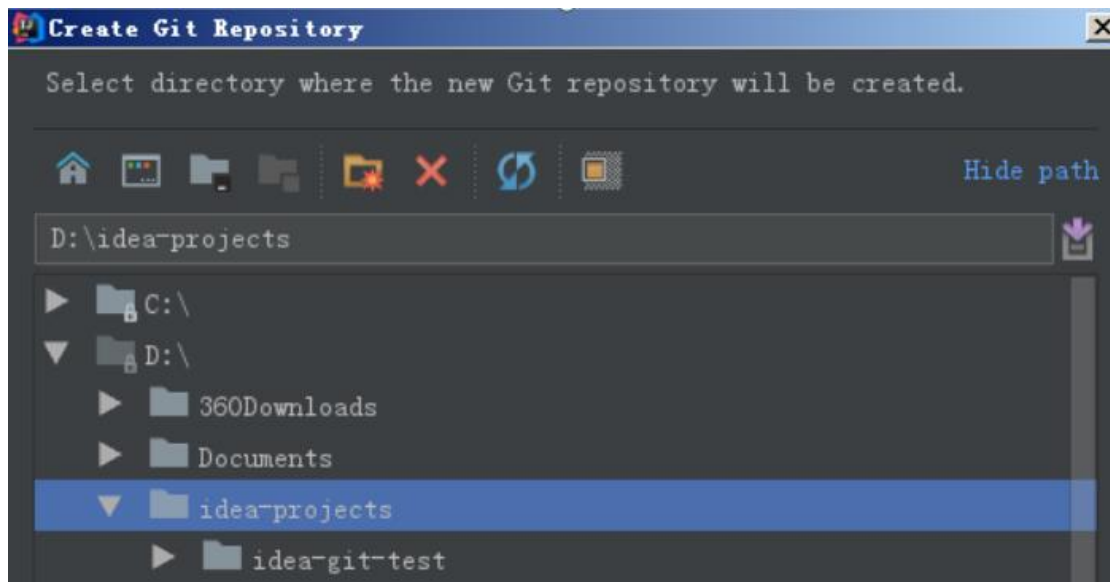
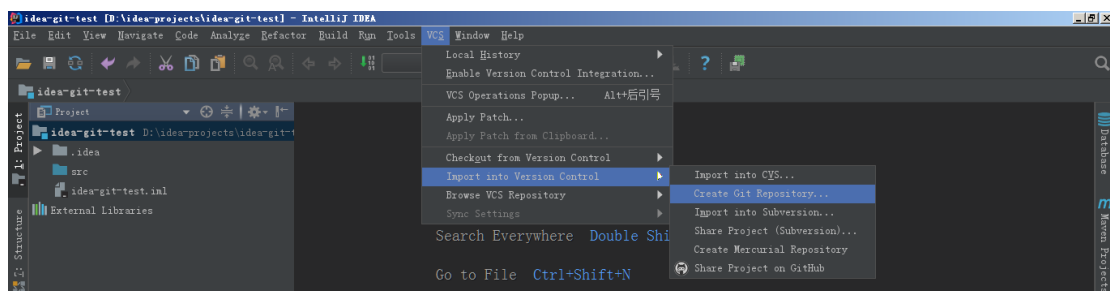
(二) 将工程添加至 git

1) 在 idea 中创建一个工程，例如创建一个 java 工程，名称为 idea-git-test，如下图所示：



2) 创建本地仓库

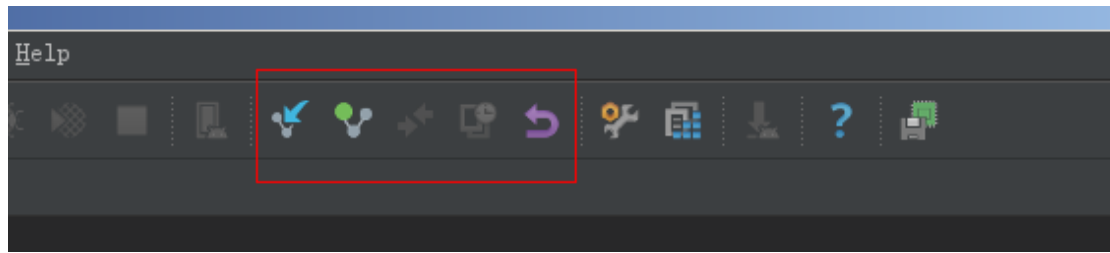
在菜单中选择“vcs”→Import into Version Control→Create Git Repository...



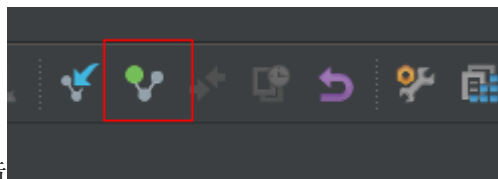
选择工程所在的上级目录。本例中应该选择 idea-projects 目录，然后点击“OK”按钮，在工程的上级目录创建本地仓库，那么 idea-projects 目录就是本地仓库的工作目录，此目录中的工程就可以添加到本地仓库中。也就是可以把 idea-git-test 工程添加到本地仓库中。

...

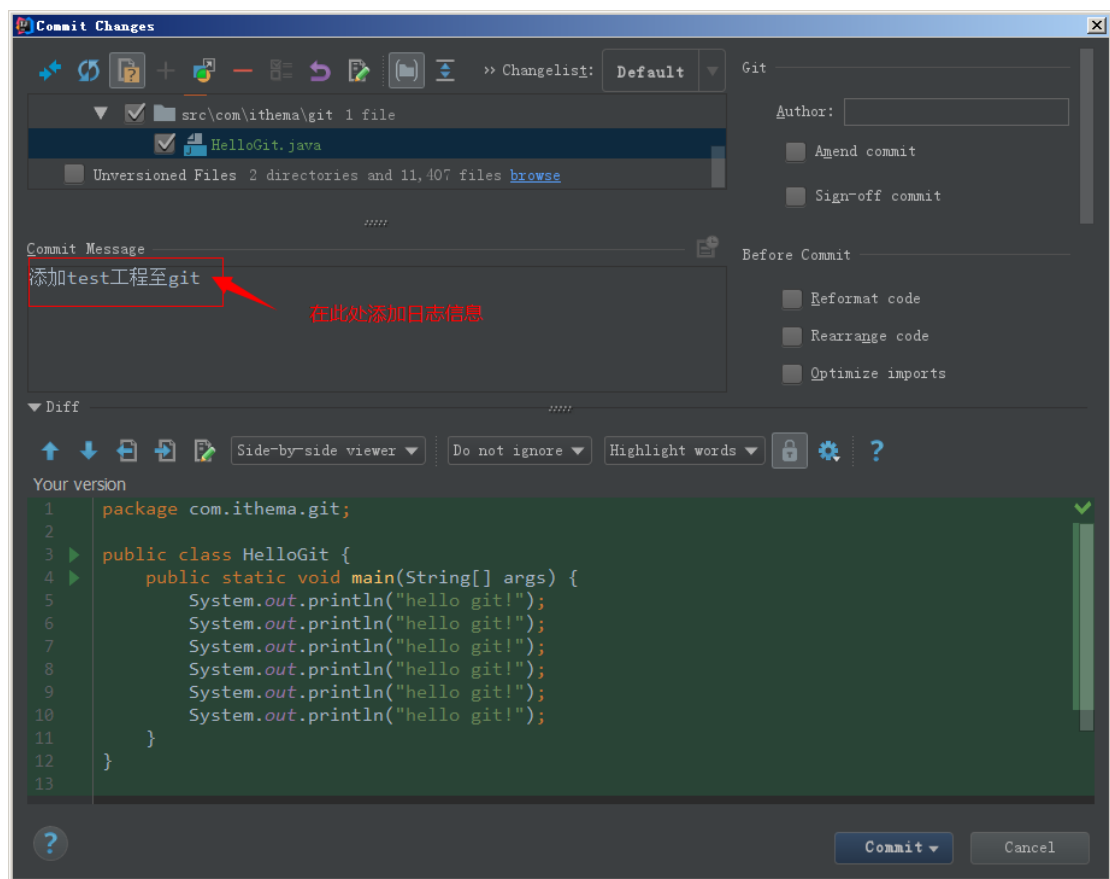
选择之后在工具栏上就多出了 git 相关工具按钮：



3) 将工程添加至本地仓库



直接点击 commit 按钮，将工程提交至本地仓库。



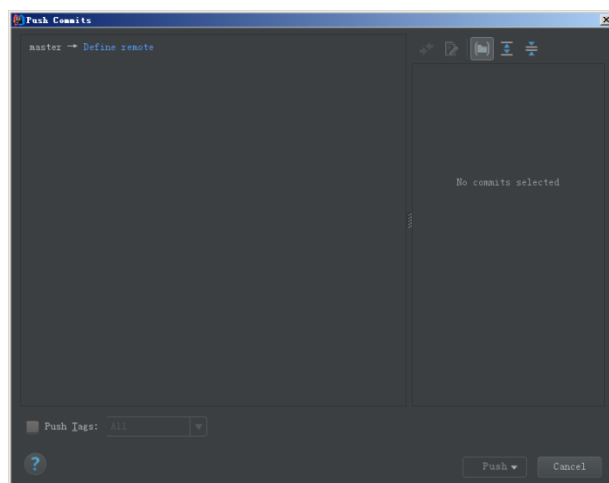
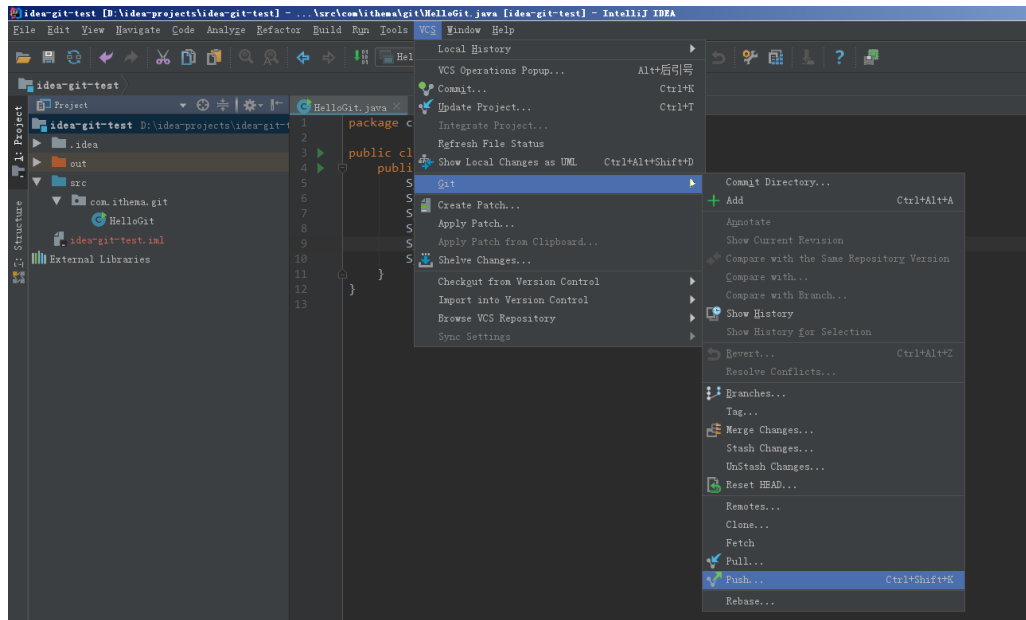
然后点击“commit”按钮，将工程添加至本地仓库。

4) 推送到远程

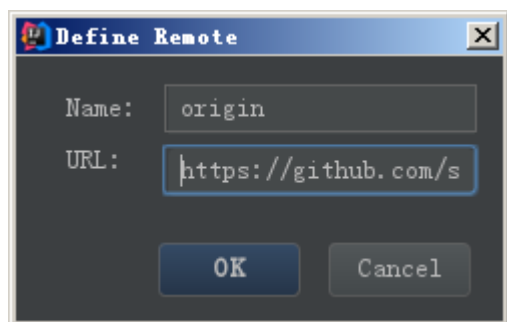
在 github 上创建一个仓库然后将本地仓库推送到远程。

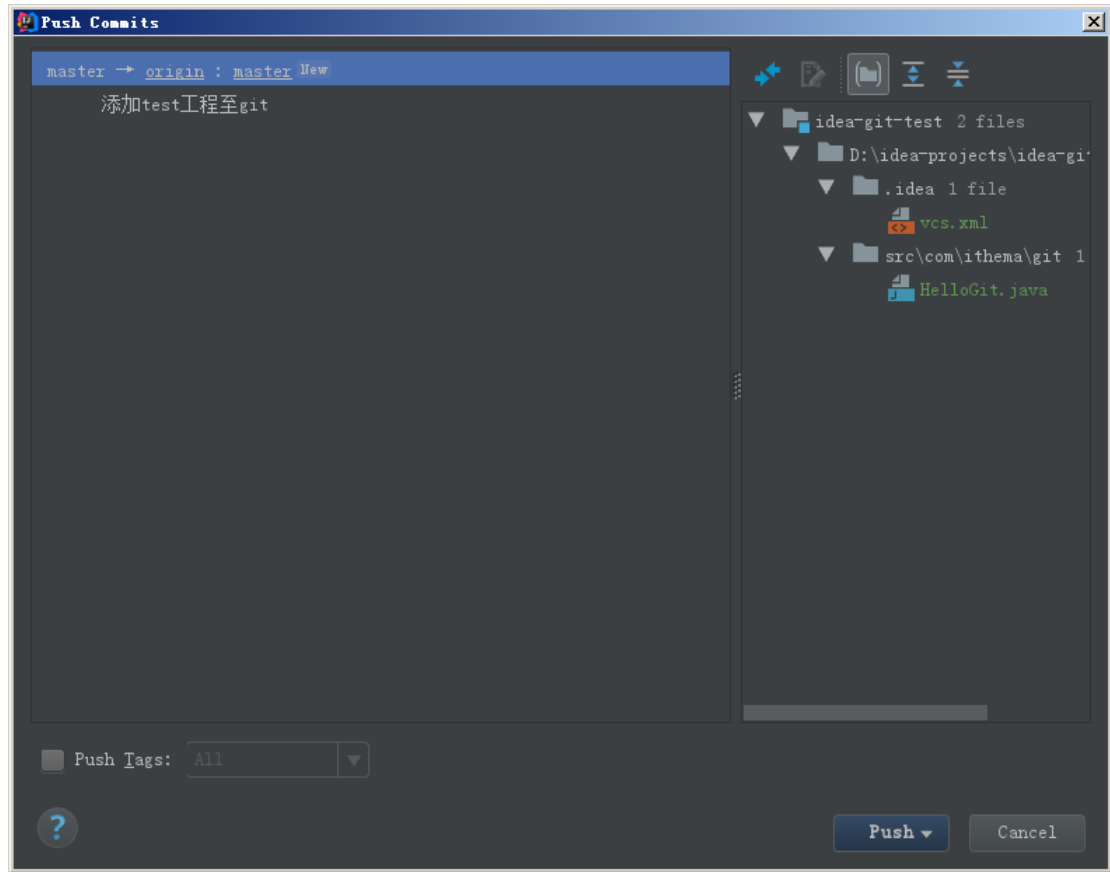
在工程上点击右键，选择 git→Repository→push，

或者在菜单中选择 vcs→git→push



点击“Define remote”链接，配置 https 形式的 URL，git 形式的无法通过。然后点击 OK

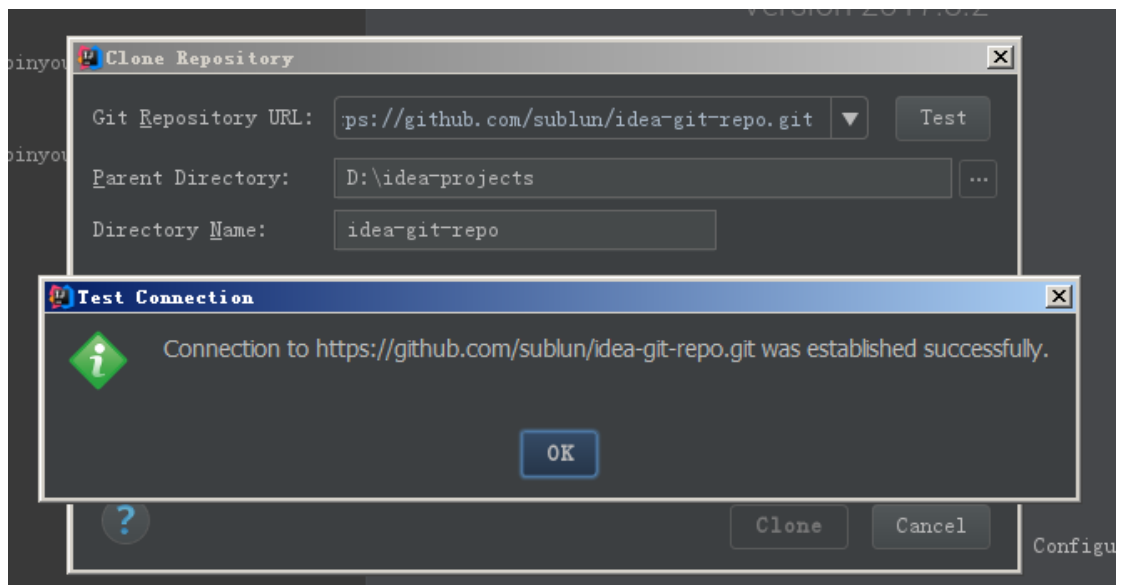




点击“push”按钮就将本地仓库推送到远程，如果是第一次配置推送需要输入 github 的用户名和密码。

(三) 从远程仓库克隆

关闭工程后，在 idea 的欢迎页上有“Check out from version control”下拉框，选择 git

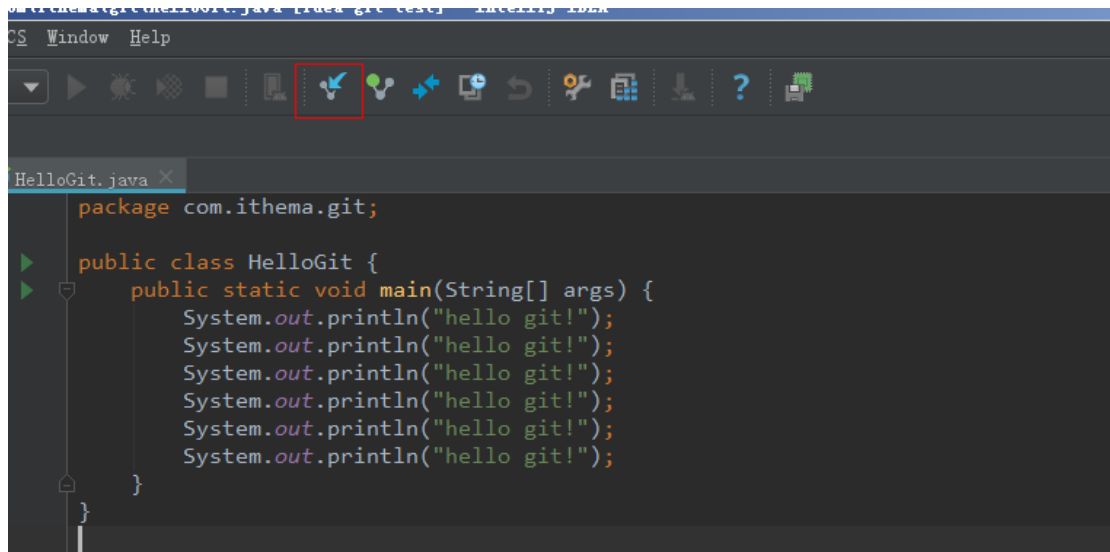


此处仍然推荐使用 https 形式的 url，点击“test”按钮后显示连接成功。

点击 OK 按钮后根据提示将远程仓库克隆下来，然后倒入到 idea 中。

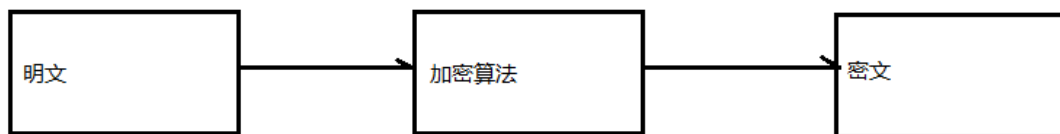
(四) 从服务端拉取代码

如果要从服务端同步代码可以使用工具条中的“update”按钮



十、Git 基本原理[了解]

(一) 哈希

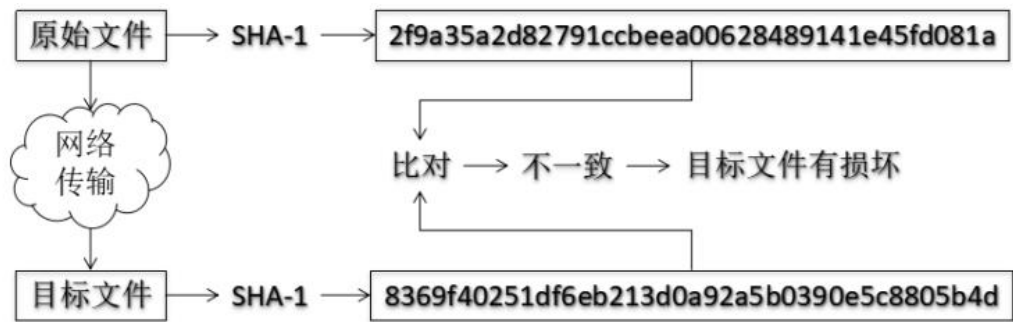


哈希是一个系列的加密算法，各个不同的哈希算法虽然加密强度不同，但是有以下几个共同点：

- 不管输入数据的数据量有多大，输入同一个哈希算法，得到的加密结果长度固定
- 哈希算法确定，输入数据确定，输出数据能够保证不变
- 哈希算法确定，输入数据有变化，输出数据一定有变化，而且通常变化很大
- 哈希算法不可逆

Git 底层采用的是 SHA-1 算法

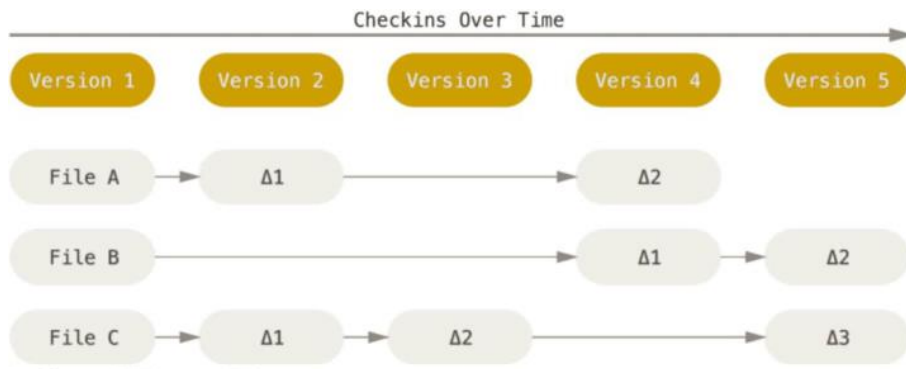
哈希算法可以被用来验证文件，原理如下图所示



Git 就是靠这种机制来从根本上保证数据完整性的。

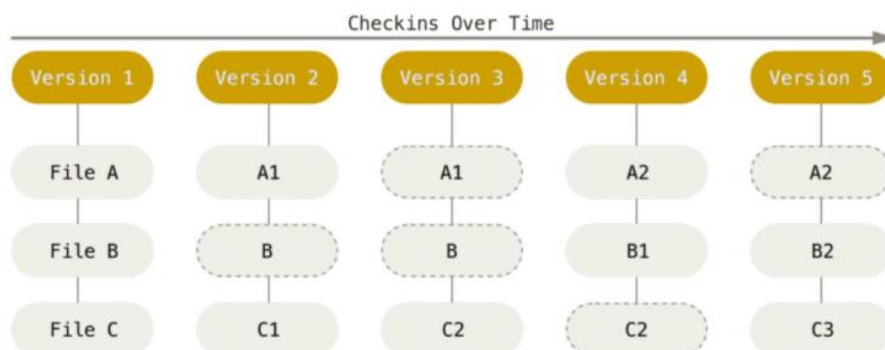
(二) Git 保存版本的机制

- 集中式版本控制工具的文件管理机制
以文件变更列表的方式存储信息。这类系统将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。



- Git 的文件管理机制

Git 把数据看作是小型文件系统的一组快照。每次提交更新时 Git 都会对当前的全部文件制作一个快照并保存这个快照的索引。为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个连接指向之前存储的文件。所以 Git 的工作方式可以称之为快照流



十一、作业

- 思考：如何做一个 Github 上开源项目的源码贡献者？
- 理解版本控制思想
- 掌握 Git 命令行操作（按照习惯流程完成一遍）
- 掌握 Idea 集成 Git 后的分享项目到 GitHub 和从 Github 检出项目
- 视频（有兴趣可观看）：作者 linus 介绍 Git 的特点和设计思路

<https://v.qq.com/x/page/o0772kqh5iv.html>