Git
Git命令行操作
本地库操作
本地库初始化
设置签名
版本控制操作
基本操作
本地库、暂存区、工作区
版本的前进和后退
查看历史版本记录
基于索引值前进后退版本（推荐方式）
reset命令的三个参数的对比
删除文件并找回
分支操作
查看分支
创建分支
切换分支
合并分支
解决分支合并产生的冲突
将本地库内容推送到远程库
获取远程库地址
执行推送
从远程库克隆到本地库

Git

Git命令行操作

本地库操作

本地库初始化

本地库初始化说白了就是建立本地库。

命令是：git init

效果如下：

1. 在D:\study\studyGit\这个路径下，鼠标右键，点击**Git Bash Here**，打开控制台。
2. 输入**mkdir BlockChain**命令，新建一个BlockChain文件夹，让这个目录代表我们当前的项目，开发这个项目需要用Git帮助我们去做版本控制。
3. 用**cd BlockChain**命令进入这个目录，现在用**ls -lA**命令查看该目录，该命令查看当前目录下所有文件和目录，包括隐藏的文件和目录，发现这个目录是空的（当然是空的）。
4. 在BlockChain目录下执行**git init**命令，初始化git本地库，在控制台里得到*Initialized empty Git repository in D:/study/studyGit/BlockChain/.git/*这个提示，意思是在*D:/study/studyGit/BlockChain/.git/*这个路径下初始化了一个空的Git仓库，*.git*目录在Linux里面是隐藏的资源，只要是以点开头命名的文件在Linux里面都是隐藏资源，初始化后的效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ ll .git
total 7
-rw-r--r-- 1 think 197121 130 12月 13 19:37 config
-rw-r--r-- 1 think 197121  73 12月 13 19:37 description
-rw-r--r-- 1 think 197121  23 12月 13 19:37 HEAD
drwxr-xr-x 1 think 197121   0 12月 13 19:37 hooks/
drwxr-xr-x 1 think 197121   0 12月 13 19:37 info/
drwxr-xr-x 1 think 197121   0 12月 13 19:37 objects/
drwxr-xr-x 1 think 197121   0 12月 13 19:37 refs/
```

注意：

.git中存放的是本地库相关的子目录和文件，不能删除也不能胡乱修改。

设置签名

形式：

- 用户名：tom
- Email地址：superMan@163.com

  Email地址不必非要和用户名一致，而且就算Email地址不存在都没关系，设置这个的目的就是为了标识开发人员的身份，能把开发人员的身份区分开就行了。

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。命令是：**git config**

```
git config user.name tom_pro
git config user.email superMan_pro@163.com
// pro 就是 project
```

- 系统用户级别：登录当前电脑操作系统的用户范围，比项目级别范围要大。命令是：**git config --global**

```
git config --global user.name tom_glb
git config --global user.email superMan_glb@163.com
// glb 就是 global
```

两种级别的优先级问题：如果这两个级别都设置了，那么项目级别/仓库级别的签名会生效，系统用户级别的签名就不会生效。

那么设置好项目级别/仓库级别的签名之后，在哪里能看到设置的用户名和email呢？用**cat .git/config**命令，打开.git文件夹里config文件，可以看到，如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat .git/config
[core]
        repositoryformatversion = 0
        filemode = false
        bare = false
        logallrefupdates = true
        symlinks = false
        ignorecase = true
[user]
        name = tom_pro
        email = superMan_pro@163.com

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

上面设置的是项目级别/仓库级别的用户名和email，下面是设置系统用户级别的用户名和email，设置完之后，在哪里能看到呢？

1. 用cd ~命令进入系统Home路径下
2. 用ls -lA命令查看Home目录里的隐藏文件和目录
3. 可以看到.*gitconfig*这个隐藏文件
4. 用cat .gitconfig这个命令可以看到这个文件中的内容，如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 ~
$ cat .gitconfig
[user]
        name = tom_glb
        email = superMan_glb@163.com

think@DESKTOP-6R37OA0 MINGW64 ~
```

实际开发中，设置一个系统用户级别就够了。

版本控制操作

- 查看工作区的状态git status

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

  On branch master 表示在master这个分支上

  No commits yet表示现在还没有任何的提交

  nothing to commit代表也没有什么可提交的，可提交的都放在暂存区，也就代表着暂存区里也没有东西

- 使用vim test.txt在工作区里新建一个文件

  然后按i键在txt里输入东西，可以随便输入一些内容，然后按Esc键，再按shift加:键（一定要是英文输入法模式下），在冒号后面输入wq后按回车键保存退出

- 新建好test.txt文件后，再用git status命令查看

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master

No commits yet

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

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

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

On branch master与No commits yet不用再解释了，Untracked files表示未追踪的文件，这里未追踪的文件用红色标识出来了--test.txt，以后我们大概就可以判断用红色标出来的文件名表示该文件还没有被提交到暂存区，然后这个文件可以用**git add**命令将它添加到暂存区里。

- 用**git add test.txt**命令

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git add test.txt
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

- 再用**git status**命令查看当前工作区状态

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master

No commits yet

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


think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

文件名是绿色的，代表test.txt已经被放到暂存区里面了，控制台提示可以用命令**git rm --cached test.txt**把它从暂存区里面撤回来

- 用命令**git commit test.txt**提交文件

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   test.txt
#
~
```

在这里记录一下本次提交是要干什么事情，按**shift**加**:**键，在冒号后面输入**set nu**，按回车，可以发现行号显示出来了，能显示行号就说明我们现在在vim编辑器里，按**i**键进入编辑模式，输入自己想输入的内容，我输入的是*My first commit. New file test.txt*，然后按**Esc**键，再按**shift**加**:**键（一定要是英文输入法模式下），在冒号后面输入**wq**后按回车键保存退出

以上就提交完了，看一下提交结果，如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git commit test.txt
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory
[master (root-commit) 4e55d1e] My first commit. New file test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

warning不用管，因为是第一次提交，所以显示*root-commit*（根提交），后面跟着的4e55d1e这串数字，目前可以粗略的认为这次提交形成的版本号（后面就会知道这是一串哈希值的一部分），后面跟着的内容，就是刚才我自己写的内容，然后显示一个文件被修改了，*1 insertion(+)*代表test.txt文件里面有1行内容，增加了一行

- 再用命令**git status**查看工作区状态

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master
nothing to commit, working tree clean

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

nothing to commit表示暂存区里没有东西可提交，working tree clean表示工作区也没有文件被修改或者新建

- 用命令**vim test.txt**修改文件

  在test.txt文件里插入一行*Something changes*，然后保存退出

- 再用命令**git status**查看工作区状态

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (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:   test.txt

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

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

*modified test.txt*代表检测到test.txt文件被修改，Changes not staged for commit代表修改没有被暂存，git restore test.txt用来在工作区取消修改，等涉及到历史记录再说，"git add" and/or "git commit -a"表示可以用**git add**命令+**git commit -a**命令修改提交，或者直接使用命令**git commit -a**提交修改

- 用命令**git add test.txt**和命令**git commit -m "My second commit, modify test.txt" test.txt**

```
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory
[master 5e2f4a4] My second commit, modify test.txt
 1 file changed, 1 insertion(+)
```

基本操作

- 状态查看操作：git status

  查看工作区和暂存区的状态

- 添加操作：git add [file name]

  将工作区文件的“新建/修改”添加到暂存区

- 提交操作：git commit -m "修改的消息" [file name]

  将暂存区的内容提交到本地库

本地库、暂存区、工作区

- 工作区就是写代码的地方
- 暂存区是一个临时存储区，存在这里面的东西可以撤掉
- 本地库是存放历史版本的

版本的前进和后退

查看历史版本记录

在对版本进行操作之前，先对版本记录有一个直观的感受，用git log命令查看历史版本记录，如下图所示：

```
think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
$ git log
commit 5e2f4a402cbc795a4d8909d4770a391baada7db3 (HEAD -> master)
Author: tom_pro <superMan_pro@163.com>
Date:   Sun Dec 13 22:53:32 2020 +0800

    My second commit, modify test.txt

commit 4e55d1ed0caff4c7bc2e45ba3838d00789d22938
Author: tom_pro <superMan_pro@163.com>
Date:   Sun Dec 13 21:50:31 2020 +0800

    My first commit. New file test.txt

think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
```

commit后面跟的一串数字是哈希值，代表本次提交的一个索引，后面的Head指向master代表指向当前的版本，版本的前进后退就是移动Head这个指针

git log --pretty=oneline命令可以让每个版本只用一行就显示出来，效果如下图所示：

```
think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
$ git log --pretty=oneline
7eb2670989bc6d8d495578cd810a4b693a6890ae (HEAD -> master) to see more history
5e2f4a402cbc795a4d8909d4770a391baada7db3 My second commit, modify test.txt
4e55d1ed0caff4c7bc2e45ba3838d00789d22938 My first commit. New file test.txt
```

更多的查看历史版本命令如下所示：

```
think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
$ git log --oneline
7eb2670 (HEAD -> master) to see more history
5e2f4a4 My second commit, modify test.txt
4e55d1e My first commit. New file test.txt

think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
```

```
think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
$ git reflog
7eb2670 (HEAD -> master) HEAD@{0}: commit: to see more history
5e2f4a4 HEAD@{1}: commit: My second commit, modify test.txt
4e55d1e HEAD@{2}: commit (initial): My first commit. New file test.txt

think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
```

git reflog命令对于指针移动非常具有参考价值

基于索引值前进后退版本（推荐方式）

命令：

git reset --hard [版本索引值]

先查看以前的版本记录：

```
think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
$ git reflog
e75946b (HEAD -> master) HEAD@{0}: commit: more changes to produce history
7eb2670 HEAD@{1}: commit: to see more history
5e2f4a4 HEAD@{2}: commit: My second commit, modify test.txt
4e55d1e HEAD@{3}: commit (initial): My first commit. New file test.txt

think@DESKTOP-6R37OAO MINGW64 /d/study/studyGit/BlockChain (master)
```

再查看文件里的内容：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat test.txt
I am new hand
Something changes
test history
2020 12 14

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

发现文件里的内容每一行从上到下都代表了每个版本对应的修改的内容

回退到版本*5e2f4a4*：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reset --hard 5e2f4a4
HEAD is now at 5e2f4a4 My second commit, modify test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

再查看文件里的内容：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat test.txt
I am new hand
Something changes

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

发现*5e2f4a4*版本之后修改的内容都消失了

这时我们查看版本记录：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reflog
5e2f4a4 (HEAD -> master) HEAD@{0}: reset: moving to 5e2f4a4
e75946b HEAD@{1}: commit: more changes to produce history
7eb2670 HEAD@{2}: commit: to see more history
5e2f4a4 (HEAD -> master) HEAD@{3}: commit: My second commit, modify test.txt
4e55d1e HEAD@{4}: commit (initial): My first commit. New file test.txt
```

现在再回到之前最新的版本：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reset --hard e75946b
HEAD is now at e75946b more changes to produce history

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

然后我们查看文件内容：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat test.txt
I am new hand
Something changes
test history
2020 12 14

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

之前回退版本导致消失的那些内容又回来了！

这时我们再次查看版本记录：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reflog
e75946b (HEAD -> master) HEAD@{0}: reset: moving to e75946b
5e2f4a4 HEAD@{1}: reset: moving to 5e2f4a4
5e2f4a4 HEAD@{2}: reset: moving to 5e2f4a4
e75946b (HEAD -> master) HEAD@{3}: commit: more changes to produce history
7eb2670 HEAD@{4}: commit: to see more history
5e2f4a4 HEAD@{5}: commit: My second commit, modify test.txt
4e55d1e HEAD@{6}: commit (initial): My first commit. New file test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

发现最上面三行显示的是版本更换的记录，有两行都是*reset: moving to 5e2f4a4*是因为我之前想回到之前最新版本的时候，版本号输错了。。。

reset命令的三个参数的对比

- --soft

不会碰暂存区和工作区，但是会碰本地库，在本地库移动指向版本的指针，试验一下，就是会发现文件里的内容没有消失，但是提示暂存区里的内容被修改了，但事实上暂存区里的内容没有被修改，仅仅是因为本地库里的版本被修改了，就凸现出了暂存区里的内容被修改了一样。效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reset --soft 5e2f4a4

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test.txt


think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat test.txt
I am new hand
Something changes
test history
2020 12 14

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

这里我再用**--soft**命令回退到修改之前的版本，会发现因为本地库的版本变回之前的样子了，所以暂存区相对来说就又不存在修改了，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reset --soft e75946b

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git status
On branch master
nothing to commit, working tree clean

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

- --mixed

重新设置暂存区，但是不会碰工作区，在本地库移动Head指针，另外会重置暂存区，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git reset --mixed 5e2f4a4
Unstaged changes after reset:
M       test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (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:   test.txt

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

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ cat test.txt
I am new hand
Something changes
test history
2020 12 14

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

- --hard

在本地库移动Head指针，重置暂存区和工作区

删除文件并找回

1. 另外新建一个文件del.txt，在里面随便写点东西
2. 删除文件一般是在把文件提交到本地库后再删除工作区的文件，所以第二步就是先把刚刚新建的文件提交到本地库
3. 删除文件，用**rm del.txt**命令执行删除操作，利用**git status**命令查看状态，发现del.txt文件处在deleted状态，需要把删除这个操作提交到暂存区
4. 执行**git add del.txt**命令，再执行**git status**命令，发现已经提交到暂存区了
5. 执行**git commit -m "delete del.txt" del.txt**命令，就是在本地库有一个确定的记录，记录del.txt文件已经被删除，只要本地库记录了，那么这条记录永远不会被删除
6. 执行**git reflog**命令，会看到最新版本里del.txt文件被删除了，但是在之前一个版本，del.txt文件还没有被删除，那么利用之前所学的知识，将版本回退到之前的那个版本，被删除的文件就找回来了

分支操作

查看分支

命令：

使用 命令**git branch -v**可以查看所有分支，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git branch -v
* master 57dbffb new del.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

创建分支

命令：

使用命令**git branch 自定义分支名**创建分支，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git branch hot_fix

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git branch -v
  hot_fix 57dbffb new del.txt
* master  57dbffb new del.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

切换分支

命令：

使用**git checkout 分支名**切换分支，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git checkout hot_fix
Switched to branch 'hot_fix'

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
```

- 在hot_fix分支上修改test.txt文件，具体步骤如下：
  1. 使用**vim test.txt**命令，编辑test.txt文件，在文件最后一行加入*2020 12 15 edit by hot_fix*
  2. 使用**git add test.txt**命令将修改后的文件提交到暂存区
  3. 使用**git commit -m "test branch hot_fix" test.txt**命令将修改后的文件上传到本地库

  效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ vim test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ git add test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ git commit -m "test branch hot_fix" test.txt
[hot_fix cb7e0f7] test branch hot_fix
 1 file changed, 1 insertion(+)

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ git branch -v
* hot_fix cb7e0f7 test branch hot_fix
  master  57dbffb new del.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
```

合并分支

1. 切换到接受修改的分支上，刚才我们在hot_fix分支上对test.txt文件做了修改，现在我们想把这个修改覆盖到master分支上，所以我们现在先切换到master分支上，其实我们不妨在切换到master分支后，查看一下master分支下test.txt文件中的内容，发现确实没有被修改，那么我们真的就需要把在hot_fix分支上对test.txt文件的修改添加到master分支上了
2. 执行**git merge 分支名**命令

具体步骤如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ git checkout master
Switched to branch 'master'

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ vim test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
$ git merge hot_fix
Updating 57dbffb..cb7e0f7
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
```

解决分支合并产生的冲突

- 产生冲突原因：

  比如在master和hot_fix两个分支上，我们可以同时推进项目的进度，当我们同时在两个分支上的同一个地方进行修改的话，比如都在test.txt文件的第六行插入了一串不一样的字符，Git在合并时就拿不定注意了，这时就会由我们人来决定怎么修改。

- 演示过程如下：

  1. 在master分支下，修改test.txt文件的第五行，改为*2020 12 15 edit by master*，然后保存退出
  2. 然后提交到暂存区，再提交到本地库，具体看下图：

     ```
     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
     $ vim test.txt

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
     $ git add test.txt

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
     $ git commit -m "test conflict by master" test.txt
     [master 31a1225] test conflict by master
      1 file changed, 1 insertion(+), 1 deletion(-)

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
     ```

  3. 切换到hot_fix分支下，修改test.txt文件的第五行，改为*2020 12 15 edit by hot_fix*，然后保存退出
  4. 然后提交到暂存区，再提交到本地库，具体看下图：

     ```
     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (master)
     $ git checkout hot_fix
     Switched to branch 'hot_fix'

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
     $ vim test.txt

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
     $ git add test.txt

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
     $ git commit -m "test conflict by hot_fix" test.txt
     On branch hot_fix
     nothing to commit, working tree clean

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
     $ git branch -v
     * hot_fix cb7e0f7 test branch hot_fix
       master  31a1225 test conflict by master

     think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
     ```

     可以看到现在两个分支的版本都不一样了

  5. 现在执行合并操作，在hot_fix分支上，把master分支合并过来，我们看一下，会发生什么：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
$ git merge master
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
```

发生了冲突！文件自动合并失败了，后面显示hot_fix|MERGING表示hot_fix分支正处在合并状态下

- 解决冲突：

1. 进入到冲突文件里，编辑文件到自己满意的状态，先说一下冲突文件里是什么样子的，如下图所示：

```
1 I am new hand
2 Something changes
3 test history
4 2020 12 14
5 <<<<<<< HEAD
6 2020 12 15 edit by hot_fix
7 =======
8 2020 12 15 edit by master
9 >>>>>>> master
```

蓝框与红框里分别指示出了冲突内容，并且还指出了是master分支与当前分支起冲突了，现在我们就来修改这个文件到我们满意的状态

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
$ git status
On branch hot_fix
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:   test.txt

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

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
```

修改完后，保存退出，执行**git status**命令，查看工作区状态，提示我们可以用**git add <file>**命令去标记这个要去解决冲突的文件

2. 可以用**git add <file>**命令去标记这个要去解决冲突的文件，然后我们再用**git status**命令查看状态，提示我们用**git commit**命令去完成合并操作，记住，此时**git commit**命令后面不能带文件名，但是必要的日志信息还是需要带上的

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
$ git add test.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
$ git status
On branch hot_fix
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   test.txt


think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
```

3. 执行**git commit**命令去完成合并操作，效果如下图所示：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix|MERGING)
$ git commit -m "resolve merge conflict by hot_fix"
[hot_fix d129573] resolve merge conflict by hot_fix

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/BlockChain (hot_fix)
```

发现原本后面显示的MERGING就消失了

将本地库内容推送到远程库

获取远程库地址

1. 在GitHub上创建一个仓库，然后获取该仓库的地址，如图所示：

具体做法就是进入到该仓库里，点击黄色按钮，复制地址：*https://github.com/IridescentFirmament/JavaStudy.git*

2. 存储远程库地址

每次都弄一个这么长的地址着实麻烦，Git提供了一种方法来存储这个地址

命令：**git remote add 地址别名 具体地址**

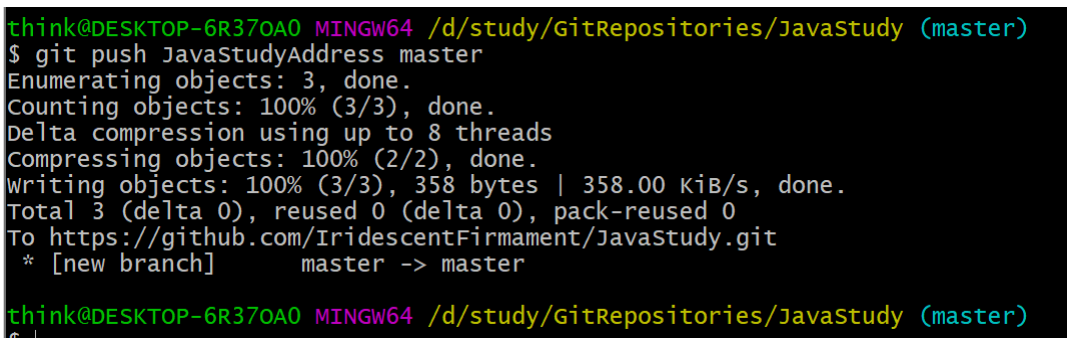
具体操作见下图：

其中**git remote -v**命令可以查看远程库地址别名和具体地址是啥，fetch表示这个地址用来取回，push设个地址表示用来推送

执行推送

命令：**git push 地址别名 想要推送的分支**

具体如下图所示：



推送过程可能会弹出一个对话框让你填GitHub的账号密码，如果浏览器上已经登录了，那么对话框就不会弹出来了，由于是远程推送，所以推送时间会长一些

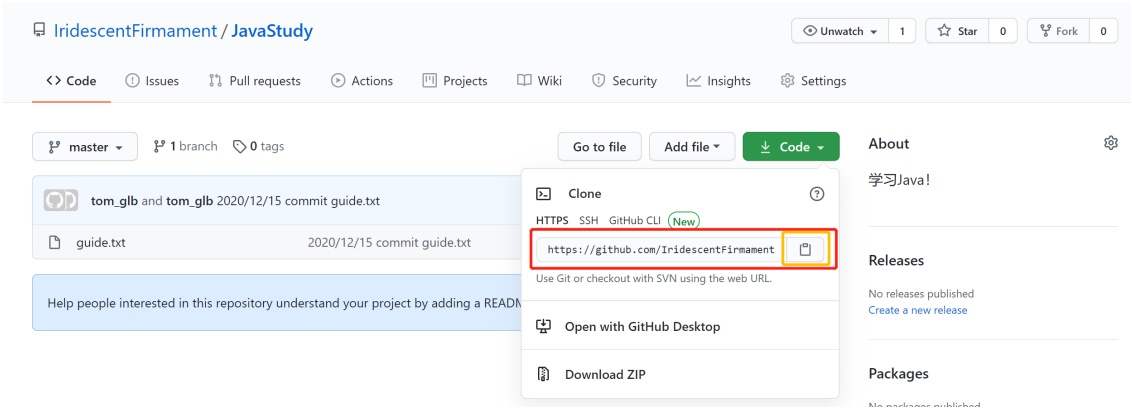
从远程库克隆到本地库

1. 在本地建立一个文件夹，用来存放从远程库克隆下来的文件

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit
$ mkdir cloneFile

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit
```

2. 获取远程库的地址



3. 执行克隆命令：`git clone 远程库地址`

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile
$ git clone https://github.com/IridescentFirmament/JavaStudy.git
Cloning into 'JavaStudy'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 338 bytes | 30.00 KiB/s, done.

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile
```

4. 一个克隆命令为我们干了很多事：

```
think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile
$ cd JavaStudy/

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile/JavaStudy (master)
$ ll
total 1
-rw-r--r-- 1 think 197121 144 12月 15 12:16 guide.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile/JavaStudy (master)
$ vim guide.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile/JavaStudy (master)
$ ls -lA
total 5
drwxr-xr-x 1 think 197121   0 12月 15 12:16 .git/
-rw-r--r-- 1 think 197121 144 12月 15 12:18 guide.txt

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile/JavaStudy (master)
$ git remote -v
origin  https://github.com/IridescentFirmament/JavaStudy.git (fetch)
origin  https://github.com/IridescentFirmament/JavaStudy.git (push)

think@DESKTOP-6R37OA0 MINGW64 /d/study/studyGit/cloneFile/JavaStudy (master)
```

首先为我们拉去远程库，然后生成了一个.git文件，为我们初始化了本地库，而且还为我们创建了远程地址别名