# Git学习笔记

[Git学习笔记](#3695-1555642024889)

[Git简介](#3460-1555642739482)

[集中式vs分布式：](#9571-1555643035032)

[Git安装](#8358-1555645515825)

[创建版本库](#6275-1555645510890)

[时光机穿梭](#6799-1570525507676)

[远程仓库](#2089-1555686444921)

[准备工作](#4220-1555750704065)

[添加远程库](#8451-1555686770214)

[修改远程库](#1069-1562118385509)

[SSH警告](#6550-1562118291272)

[从远程仓库克隆](#5015-1555770184133)

[分支管理](#4025-1555770751069)

[标签管理](#6568-1556325237623)

[使用GitHub](#1088-1556374549604)

关于教程作者

[廖雪峰](http://weibo.com/liaoxuefeng)，十年软件开发经验，业余产品经理，精通Java/Python/Ruby/Visual Basic/Objective C等，对开源框架有深入研究，著有《Spring 2.0核心技术与最佳实践》一书，多个业余开源项目托管在[GitHub](https://github.com/michaelliao)，官网：<https://www.liaoxuefeng.com>

## Git简介

Git是世界上最先进的分布式版本控制系统。

Git是linux的创建者Linus花了2周时间用C语言写的。

### 集中式vs分布式：

* 集中式（CVS及SVN）——版本控制放在中央服务器，缺点是联网后大家才能工作。
* 分布式（Git）——每个人电脑都是一个版本库，坏一台半台电脑不影响版本控制，安全性高。

## 

## Git安装

安装完成后，用以下命令将用户信息全局初始化。

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

## 

## 创建版本库

$ mkdir learngit $ cd learngit $ pwd

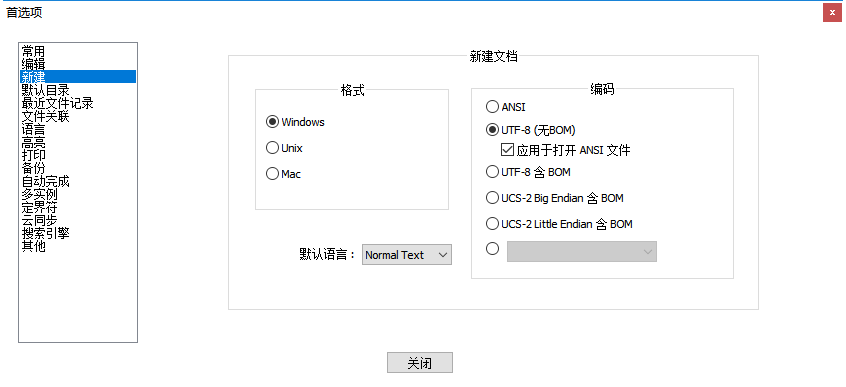
以上三行命令分别为创建目录、进入目录以及查看当前目录位置。

注意：路径中最好不要有中文

$ git init

初始化，会在此目录下建立一个隐藏的.git文件夹，用来跟踪管理版本库。

windows下编辑文本文件时最好用[Notepad++](http://notepad-plus-plus.org/)而不要用自带记事本，因为后者在文件头部添加的十六进制字符0xefbbbf容易导致编译报错。Notepad++最好设置默认编码为UTF-8 without BOM。



$ git add readme.txt

添加文件，可连续多次添加。

$ git **commit** -m "wrote a readme file"

提交文件，把添加的所有文件一次提交，-m后面是提交信息，最好写有意义的描述。方便以后看到提交原因。

## 发现，如果提交的注释中有中文，则可能在gitBash中导致光标错位等问题。

## 时光机穿梭

1. **查看状态和版本对比**

先将原文件的第7行加入单词distributed ，然后在第8行后面添加一行Very Good!

$ git status

显示当前库的状态，会提示变动过哪些文件。

$ git diff test.txt

查看和上一版本的具体变动内容

显示内容如下：

diff --git a/test.txt b/test.txt index 629d9c8..3d98a7f 100644 --- a/test.txt +++ b/test.txt @@ -4,8 +4,9 @@ test line3. test line4. test line5. test line6. -Git is a version control system. +Git is a distributed version control system. Git is free software. +Very Good! test line7. test line8. test line9.

详解：

1. diff --git a/test.txt b/test.txt ——对比两个文件，其中a改动前，b是改动后，以git的diff格式显示；
2. index 629d9c8..3d98a7f 100644 ——两个版本的git哈希值，index区域（add之后的缓存区）的 629d9c8 对象和工作区域的 3d98a7f 对象， 100表示普通文件，644表示权限控制；
3. --- a/test.txt +++ b/test.txt ——减号表示变动前，加号表示变动后；
4. @@ -4,8 +4,9 @@ test line3. ——@@表示文件变动描述合并显示的开始和结束，一般在变动前后多显示3行，其中-+表示变动前后，逗号前是起始行位置，逗号后为从起始行往后几行。合起来就是变动前后都是从第4行开始，变动前文件往后数8行对应变动后文件往后数9行。
5. 变动内容 ——+表示增加了这一行，-表示删除了这一行，没符号表示此行没有变动。
6. **版本回退**

$ git log

用来查看最近三次提交的记录

$ git log --pretty=oneline

合并每条记录到一行

WebStorm的命令窗口，log命令可以显示最近很多次提交，但对message中的中文显示不友好。

$ git re**set** --hard HEAD^

向前回退版本，其中HEAD后面跟几个^就是往回退几个版本，如果回退100个版本，可以写成 HEAD~100 。

$ git reset --hard 07e0

向后恢复版本，首先要查找到对应版本的哈希id前4位，如果提交窗口找不到，可以使用以下命令

$ git reflog

这个命令记录了每一次版本相关的操作。

git回退的速度非常快，因为在git内部有一个指向当前版本的HEAD指针，回退到某个版本，实际上是git把指针移动指向某个版本。

1. **工作区和暂存区**

* **工作区（Working Directory）：**.git所在的目录下，除了.git之外的其他文件都是在工作区内。
* **版本库（Repository）：**.git目录内所存的记录，有暂存区和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。
* **stage（或者叫index）的暂存区：**用add命令放进来文件的位置。

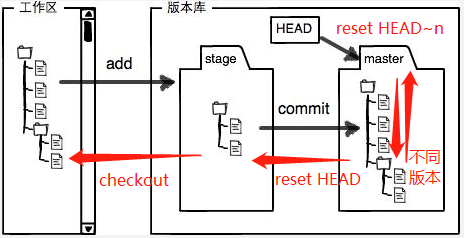
如果文件在工作区被编辑，对应的status状态就是 Changes not staged for commit

如果工作区新增文件，则对应的status状态就是 Untracked files

如果文件被add后，对应的status状态就是 Changes to be committed

多次add后的文件都放在暂存区，最后一次性全部提交。提交后的status状态就是 nothing to commit, working tree clean

这时候工作区就是干净的，暂存区也没有任何内容了。



1. **管理修改**

如果一个文件，修改一次后，add，再修改一次后直接commit，然后status则显示还有一次修改没有被提交，因为提交只对暂存区生效。所以要么每改动一次后都add，最后一次性提交；要么add一次就提交一次。

* **比较工作区与暂存区**

git diff 不加参数即默认比较工作区与暂存区

* **比较暂存区与最新本地版本库（本地库中最近一次commit的内容）**

git diff --cached  [<path>...]

* **比较工作区与最新本地版本库**

　　　　git diff HEAD [<path>...]  如果HEAD指向的是master分支，那么HEAD还可以换成master

1. **撤销修改**

**如果**在工作区修改了文件后的status，会提示，下一步可以add到暂存区，或者从暂存区恢复修改：

Changes not staged for commit: (use "git add <file>..." to update what will be committed) (use "git checkout -- <file>..." to discard changes in working directory)

想要从暂存区恢复（撤销），就用第3行的命令，其中 -- 一定不能省略：

$ git checkout -- test.txt

**如果**已经add到暂存区了，这时想要撤销操作，这时可以从status中的提示——从HEAD中恢复修改。

$ git reset HEAD

但这时候暂存区的修改撤销了，工作区还是修改后的内容，此时再使用上面提交的 $ git checkout -- test.txt 来撤销工作区修改，世界终于变得清净了！

如果已经commit，就用前面第2节回退版本的方式来撤销修改，前提是还没push上去，否则就真的不是秘密了！

1. **删除文件**

$ rm test2.txt

此命令可以从工作区删掉文件。如果要从版本库中删除，则add后提交即可，如果是误删了，则通过

$ git checkout -- test2.txt

从版本库里恢复。

如果已经将删除提交，则像前面一样先恢复版本库，然后在checkout出要恢复的文件。

## 

## 远程仓库

### 准备工作

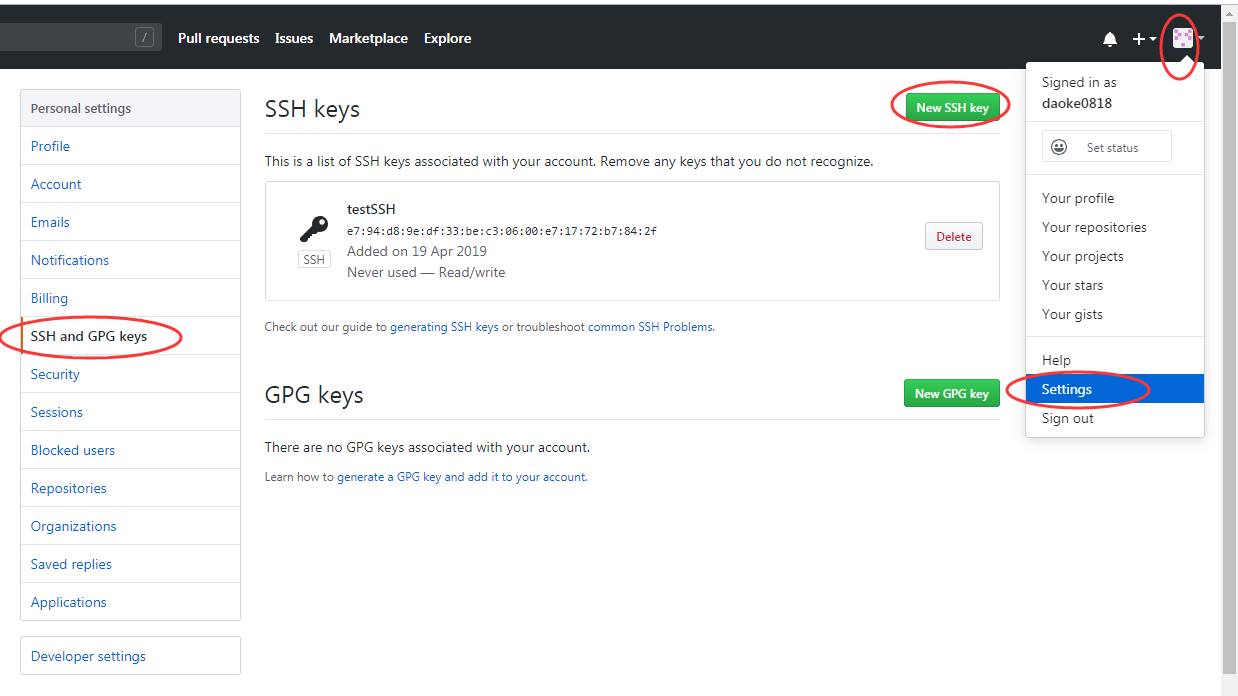
假设已经注册了github账号，开始设置：

1. 查看用户主目录（系统盘的Administrator）下有没有.ssh目录，如果有，再看里面是否有id\_rsa和id\_rsa.pub这两个文件，如果没有则需创建：

$ ssh-keygen -t rsa -C "daoke\_li@qq.com"

然后一路回车，就会自动创建这两个文件，分别是密钥对的私钥和公钥。

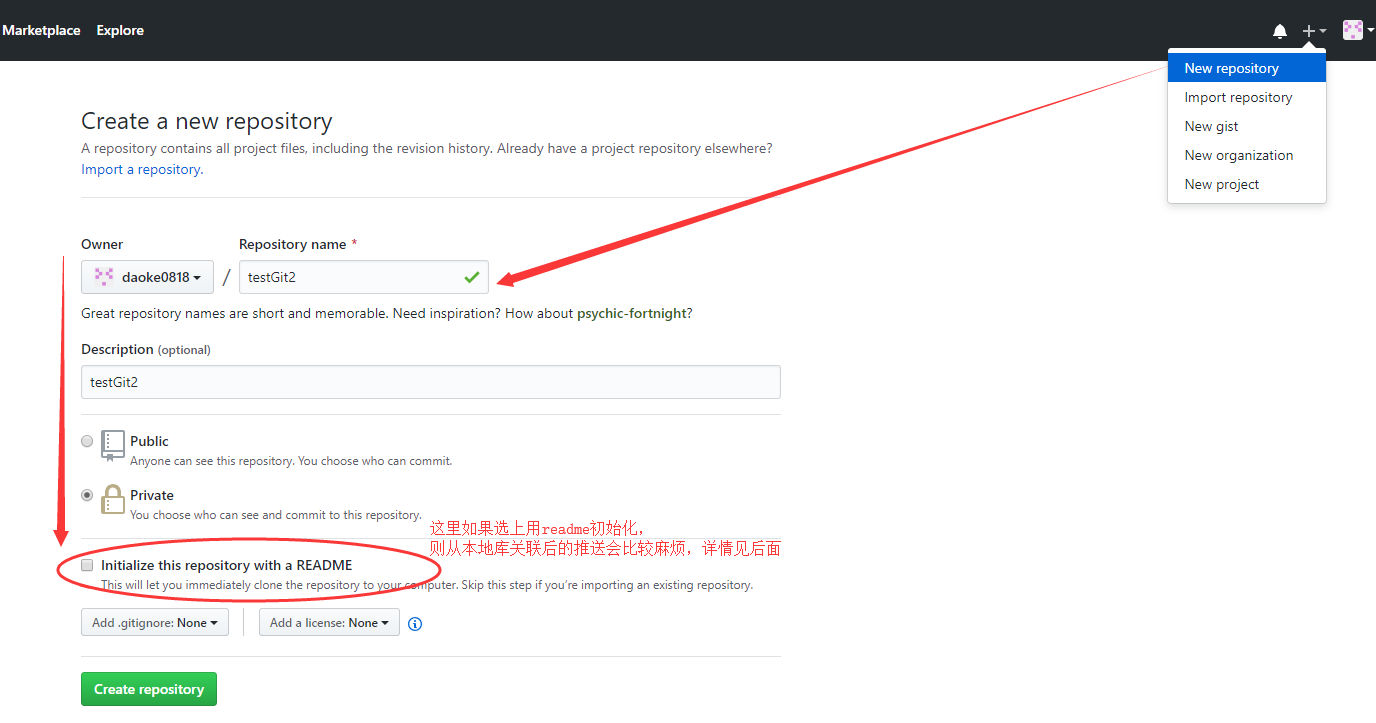
1. 在Github中添加公钥的内容：



添加公钥的目的是为了让github能识别出这台电脑，只有这台电脑才能给他推送。

### 添加远程库

1. 先在github上建一个空仓库，注意不要勾选用readme初始化。



1. 运行下面命令，让本地库与远程库关联起来

$ git remote add origin https://github.com/daoke0818/testGit2.git

廖老师博客上是通过 git@github.com:michaelliao/learngit.git 关联的，经测试推送时不能成功，https的才可以，估计原因跟网络有关系。后来在某些网络下发现git@好用，就全部用改了，果然快了很多！后来还发现以前不能成功的网络也能成功了……

如果已经用git@关联，又需要改成https协议，则在.git目录下的config文件中，把 url = 后面的内容改为https类型的即可，也可以通过后面提到的remove指令来解除原关联并重新关联或直接设置。

1. 关联成功后用如下命令把本地内容推送到远程库中，其中 -u 是指定后面的仓库名和分支名为默认，以后直接用 git push 即可：

$ git push -u origin master

如果在第一步中创建时已经初始化过项目，则这时会提醒

error: failed to push some refs to 'https://github.com/daoke0818/testGit2.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.

因为远程库中已经存在readme文件了，所以需要先pull下来。命令如下：

$ git pull origin master

这时又会报错：

From https://github.com/daoke0818/testGit \* branch master -> FETCH\_HEAD fatal: refusing to merge unrelated histories

说这两个库的git历史记录不相干而无法合并，这时我们可以加上一个参数 --allow-unrelated-histories 即可成功pull：

$ git pull origin master --allow-unrelated-histories

但是这时**可能**会提示必须输入提交的信息，默认会打开vim编辑器，先按 i 切换到插入模式，写完后 Esc→：→wq 即可保存退出编辑器。如果不进入vim编辑器，则会自动生成一个合并代码的commit。然后再使用前面的命令push将本地提交推送到远程仓库。后面如果本地还有commit，就可以直接用 git push origin master 推送。

如果需要解除关联，可以使用

$ git remote remove origin

### 

### 修改远程库

除了上面删除再添加的办法，还可以使用如下命令：

**$ git remote set-url origin git@gitee.com:daoke0818/**新的仓库地址**.git**

### SSH警告

当第一次使用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)?

回车后即把GitHub的key添加到本机的信任列表中，此警告以后不会再出现。

### 从远程仓库克隆

经测试，windows上

$ git clone git@github.com:daoke0818/testGit3.git

报错：Please make sure you have the correct access rights and the repository exists.

大概还是网络原因，用https协议就可以：

$ git clone https://github.com/daoke0818/testGit3.git

尽管ssh支持的原生git协议要比https协议快。

## 分支管理

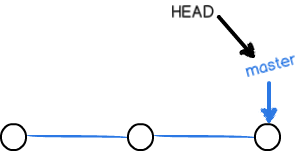
1. **概述**

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。这种情况下需要**分支**来管理。自己在创建的新分支上进行开发，完成后一次性提交合并即可。

Git对于分支的创建、切换和删除都能非常快的实现，而SVN就很慢。

1. **创建与合并分支**

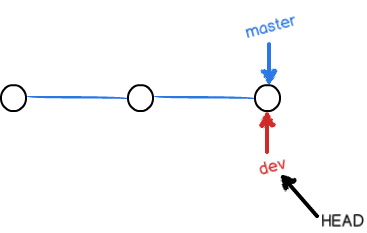
git单分支的结构是这样的，master是指向最新提交的指针，HEAD是指向master的指针，每做一次提交，指针就向前移动一步：



现在增加一个dev分支并切换到这个分支：

$ git -b checkout dev

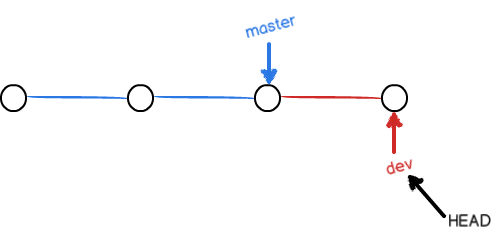
这个代码可以写成两步，分别是创建新分支 $ git branch dev ，切换到目标分支 $ git checkout dev ，之后变成这样：



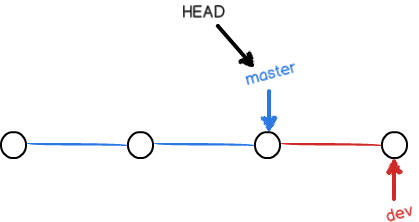
这时可以用命令查看分支

$ git branch\* dev master

其中带星号的是当前所在分支。然后在新分支上做一些更改，再add并提交，这时结构变成了这样：



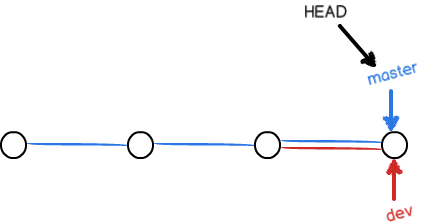
切换回master（$ git checkout master ）分支后，发现刚才所做的改动不见了，是因为改动在dev分支上。



这时使用合并命令：

$ git merge dev

即把目标分支合并到当前分支上。完成后提示 Fast-forward ，说明系统用了快进模式进行合并，此时的结构为：



master分支上也成了最新版。这时不需要dev分支了，可以删除：

$ git branch -d dev

这时再查看分支，已经没有dev了。

1. **解决冲突**

当两个分支上对同一个文件有修改并分别有提交，最后Git无法自动合并，就会产生冲突。

$ git merge 'feature2' Auto-merging test2.txt CONFLICT (content): Merge conflict in test2.txt Automatic merge failed; fix conflicts and then commit the result.

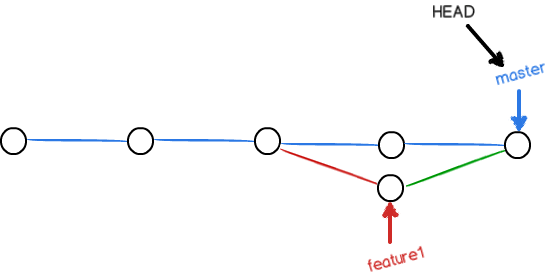
如果你不服气再执行一次合并，就会看到：

$ git merge feature2 error: Merging is not possible because you have unmerged files. hint: Fix them up in the work tree, and then use 'git add/rm <file>' hint: as appropriate to mark resolution and make a commit. fatal: Exiting because of an unresolved conflict.

这时候可以通过 git status 查看冲突信息，找到描述中冲突的文件：

<<<<<<< HEAD f4 in master ======= f4 is new >>>>>>> f4

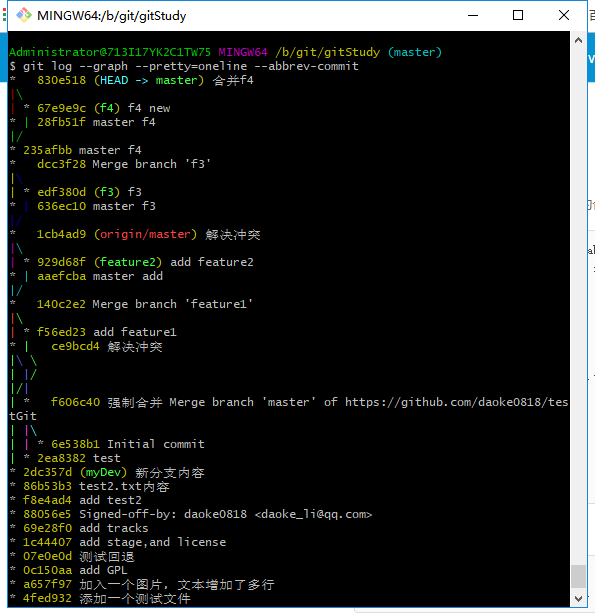
其中 <<<<<<< HEAD 和 ======= 之间是当前分支的最新版， ======= 和 >>>>>>> f4 之间是目标分支内容，手动修改后删掉这些符号，然后提交，结构如图：



可以用以下代码看到图形化流程：

$ git log --graph --pretty=oneline --abbrev-commit

其中， --graph 是图形化， --pretty=oneline 是一行显示， --abbrev-commit 是只显示每次提交id的前几位，显示效果如下：

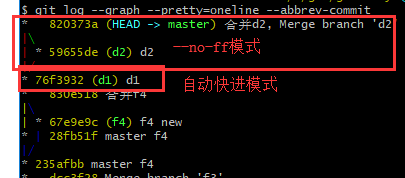


1. **分支管理策略**

默认情况下，如果情况允许，Git会自动用快进模式合并分支，但这样合并后不会留下分支存在过的痕迹。删除分支后就会丢失相应信息。如果不想这样做，则在合并时加上参数 --no-ff ，Git则会生成一个提交，所以同时再加上一个提交信息（如果不加则会进入vim模式让编辑提交信息），代码如下：

$ git merge --no-ff -m "merge with no-ff" dev

这样合并后还能看到对应的分支信息，如图，



#### 实际开发中的分支策略：

* 首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；
* 平时大家在dev分支上干活，需要发布时合并到master分支即可。

1. **Bug分支**

假设正在dev上开发，突然接到修复master上一个bug，就可以用如下命令把现场保存起来（这个命令比其他命令要执行的慢）：

$ git stash

这时工作区就是干净的，刚才的改动不见了。然后把分支切换到master，并在此基础上新建并切换到bug分支issue-101，在这里修复bug。修复完成后回到master分支，进行**非快速合并**后删除bug分支，再切换回dev分支，可以通过加list参数看到 stash 的列表：

$ git stash liststash@{0}: WIP on d3: 820373a 合并d2, Merge branch 'd2'

通过如下命令恢复现场：

$ git stash apply

这时stash区的内容还存在，可以用list查看，如果要清理掉，就用

$ git stash drop

这时stash区什么都没了。如果将工作区内容多次保存到stash，则可以加 stash@{0} 这样的编号来指定恢复哪个（可用list参数查看编号）。

$ git stash apply stash@{0}

也可以用如下命令弹出最后一次保存的工作区内容，这个命令会将对应的stash内容清除掉。

$ git stash pop

1. **Feature分支——强行删除分支**

如果在master分支上删除一个已经提交但没有合并的其它分支，则会报错：

**$ git branch -d f5error: The branch 'f5' is not fully merged.If you are sure you want to delete it, run 'git branch -D f5'.**

这时可以用参数 -D 强制删除：

$ git branch -D f5

需要注意的是，由于分支未合并，删除之后就没有任何记录了，分支上所有的修改也会丢失。

1. **多人协作**

**先来两个命令：**

$ git remote

查看远程仓库名称

$ git remote -v

查看远程仓库更详细的信息

场景：我本地有master和dev两个分支，但我只把master推送到远程仓库中。然后我的小伙伴从远程的master分支上克隆了一份，他是看不到我本地dev分支的。然后自己在本地新建dev分支进行开发，完了之后推送到远程仓库。同时我在本地 的dev分支修改了跟他一样的文件。这时我准备推送代码，就会报错，提示先pull同步代码， 但拉取的时候又报错，说没有指定本地dev和远程origin/dev之间的连接（ There is no tracking information for the current branch. ）。可通过以下代码进行关联：

$ git branch *--set-upstream-to=origin/dev dev*

然后再pull，解决冲突，再提交，再push，跟前面一样。

我在测试过程中没有出现没指定连接的报错，但其他时候出现过。

补充 ：从远程git仓库里的指定分支拉取到本地（本地不存在的分支）

git checkout -b 本地分支名 origin/远程分支名

然后再 pull

1. **Rebase**

Rebase用来整理提交记录，把多条分叉合并成一条直线。

假设有代码：

$ git log --graph --pretty=oneline --abbrev-commit \* 582d922 (HEAD -> master) add author \* 8875536 add comment \* d1be385 (origin/master) init hello \* e5e69f1 Merge branch 'dev' |\ | \* 57c53ab (origin/dev, dev) fix env conflict | |\ | | \* 7a5e5dd add env | \* | 7bd91f1 add new env ...

Git用(HEAD -> master)和(origin/master)标识出当前分支的HEAD和远程origin的位置分别是582d922 add author和d1be385 init hello，本地分支比远程分支快两个提交。假设推送时发现有人改了同样文件导致冲突，pull下来解决后再提交，这时本地分支会比远程超前1个提交。

$ git log *--graph --pretty=oneline --abbrev-commit* \* e0ea545 (HEAD -> master) Merge branch 'master' of github.com:michaelliao/learngit |\ | \* f005ed4 (origin/master) **set** exit=1 \* | 582d922 **add** author \* | 8875536 **add** comment |/ \* d1be385 init hello

如果觉得这种分叉的图形看起来乱，可以用如下命令整理一下：

$ git rebase

整理后查看到的记录为：

$ git log *--graph --pretty=oneline --abbrev-commit* \* 7e61ed4 (HEAD -> master) add author \* 3611cfe add comment \* f005ed4 (origin/master) **set** exit=1\* d1be385 init hello

发现Git把我们本地的提交“挪动”了位置，放到了f005ed4 (origin/master) set exit=1之后，这样，整个提交历史就成了一条直线。修改不再基于d1be385 init hello，而是基于f005ed4 (origin/master) set exit=1，但最后的提交7e61ed4内容是一致的。推送之后如图：

$ git log --graph --pretty=oneline --abbrev-commit \* 7e61ed4 (HEAD -> master, origin/master) add author \* 3611cfe add comment \* f005ed4 set exit=1 \* d1be385 init hello

远程和本地都成了一条直线。

Rebase的缺点是会更改我们的本地提交，但合并后的内容是一致的。

本节直接用老师博客的代码，没有手动敲。

## 标签管理

Git的标签就是版本库的快照，但其实它就是指向某个commit的指针。

1. **创建标签**

$ git tag v1.0

给当前的commit打上标签 v1.0

$ git tag

查看所有标签，经测试在dev分支上能看到master上所有标签，尽管dev上面的commit要少很多。

注意，标签不是按时间顺序列出，而是按字母排序的

$ git tag v0.9 f52c633

可以给历史commit打上标签，最后一个参数是commit的前几位（通过git log查看）

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

创建带有注释的标签，-a后面是标签名，-m后面是注释内容

$ git show v0.1

查看标签名为 v0.1的详细内容

1. **操作标签**

$ git tag -d v0.1

删除本地标签

$ git push origin v1.0

将标签 v1.0 推送到远程仓库

$ git push origin --tags

将尚未推送的标签全部推送到远程仓库

如果要删除远程仓库的标签，有以下两个步骤：

1. 删除本地标签，见上
2. 删除远程仓库对应标签

$ git push origin :refs/tags/v0.9

查看远程仓库所有标签：

git ls-remote

## 使用GitHub

* 在GitHub上，可以任意Fork开源仓库；
* 自己拥有Fork后的仓库的读写权限；
* 可以推送pull request给官方仓库来贡献代码。

connect to host gitee.com port 22: Connection timed out

Could not resolve hostname ssh.gitee.com: Name or service not known

**git config --global core.quotepath false**# 设置显示中文文件名