

## 一、git简介

git是什么，简单的说就是高效的管理代码的工具，能让你早下班的工具。

生活中我们在做一个word的时候，总会不断的在更新，但是我想回到某一个时间段的时候，怎么办，好办啊，保存-命名-保存-命名，显然你最后的word不是一个了，而是一圈了，很多很多，很抓狂的一件事，你还得慢慢的去找。当然有的小伙伴说word有定时备份和定时保存的功能，但是上班族的白领，谁会用这么高大上的东西呢，这个功能很麻烦的，用了你就会知道的。而git的功能就是很好的管理并解决你的烦恼，git会记录每一时刻你修改了什么，保存了每一个版本的状态，而且还有后悔药可以吃的，并且是多人协作。简单说就像下面的表格一样。

版本	文件名	作者	备注	时间
1	git从入门到放弃	Jingxiang	更改了标题	2020/07/19/ 14:40:13
2	git从入门到放弃	Natasha	删除了第42页内容	2020/07/20/ 10:49:28
3	git从入门到放弃	Natasha	增加了首页的批注	2020/07/20/ 23:40:46
4	git从入门到放弃	Jingxiang	首页字体改为方正	2020/07/22/ 18:50:13
5	git从入门到放弃	Ranran	替换了logo	2020/07/22/ 9:30:53

在这里，git的功能就是你想回到哪个版本就回到哪个版本，是不是很方便啊。

## 二、Git的诞生

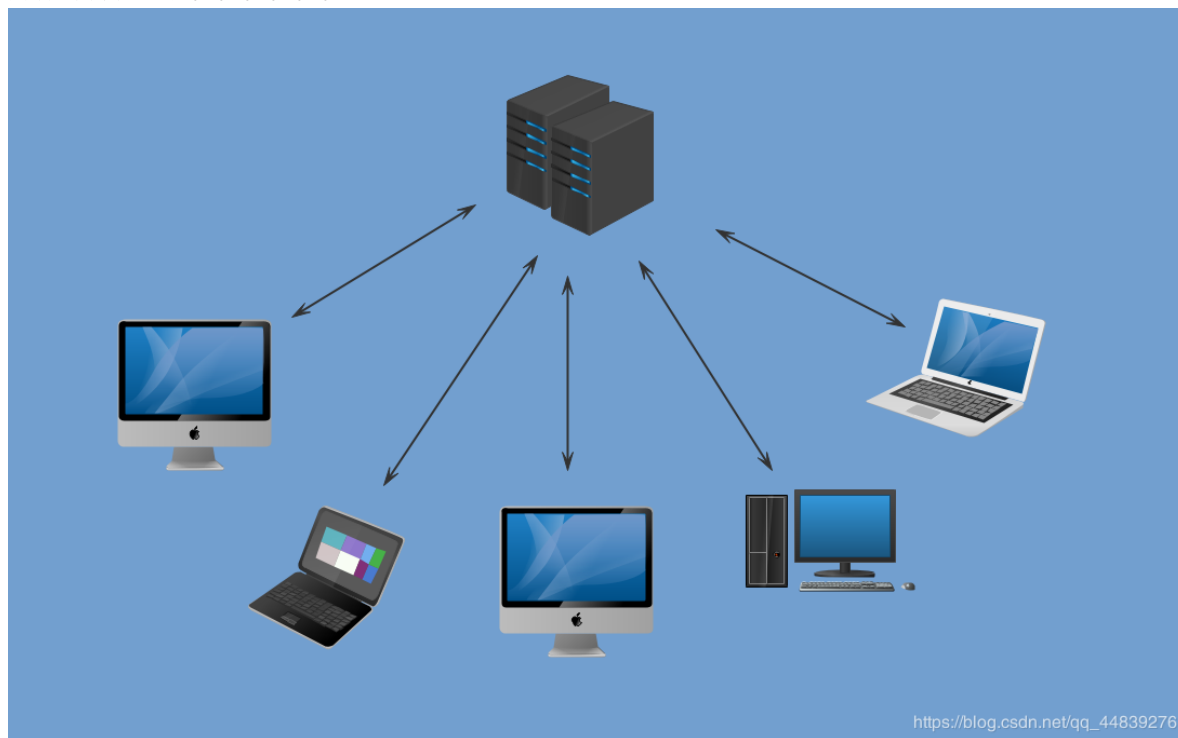
话说回来git是怎么来的呢，这就是linux这个nb人物的存在了。早些时候，Linux本着开源求知的态度，收货了一大批Linux爱好者发来的Linux代码，那个时候代码还不是很多啊，linus就一条一条的进行整理，但是随着linux发展的越来越快，很显然他自己整理不过来了，社区的小弟们也厌烦了，天天忙着整理代码，很烦躁了。于是linus为了缓解弟兄们的压力，选择了一款money的控制系统，名叫BitKeeper，开发这个控制系统的Bitmover公司看linux社区的这些人开源精神很强啊这是，于是就免费的给他们用。但是用着用着就有人不满，不满的这个人就是开发出Samba的Andrew，他想破解这个版本系统，但是最后被人家Bitmover公司发现了（其实当时有好多人的破解，但是无奈喝凉水都塞牙的Andrew被发现了），显然我给你免费用，你还要诋毁我，这可不行了，收回了，不给用了。但是linus也不是吃醋的啊，为了自己的人不受欺负，为了显示自己的能力，人家半个月时间用c自己写了一个控制系统，就是GIT了，半个月还不够，一个月之后已经正式上线运行了，这，就是先人吧。于是乎，git上线了，08年github官网上线了，可以免费的提供git存储。

### 三、集中式和分布式区别

集中式版本控制系统有CVS、SVN

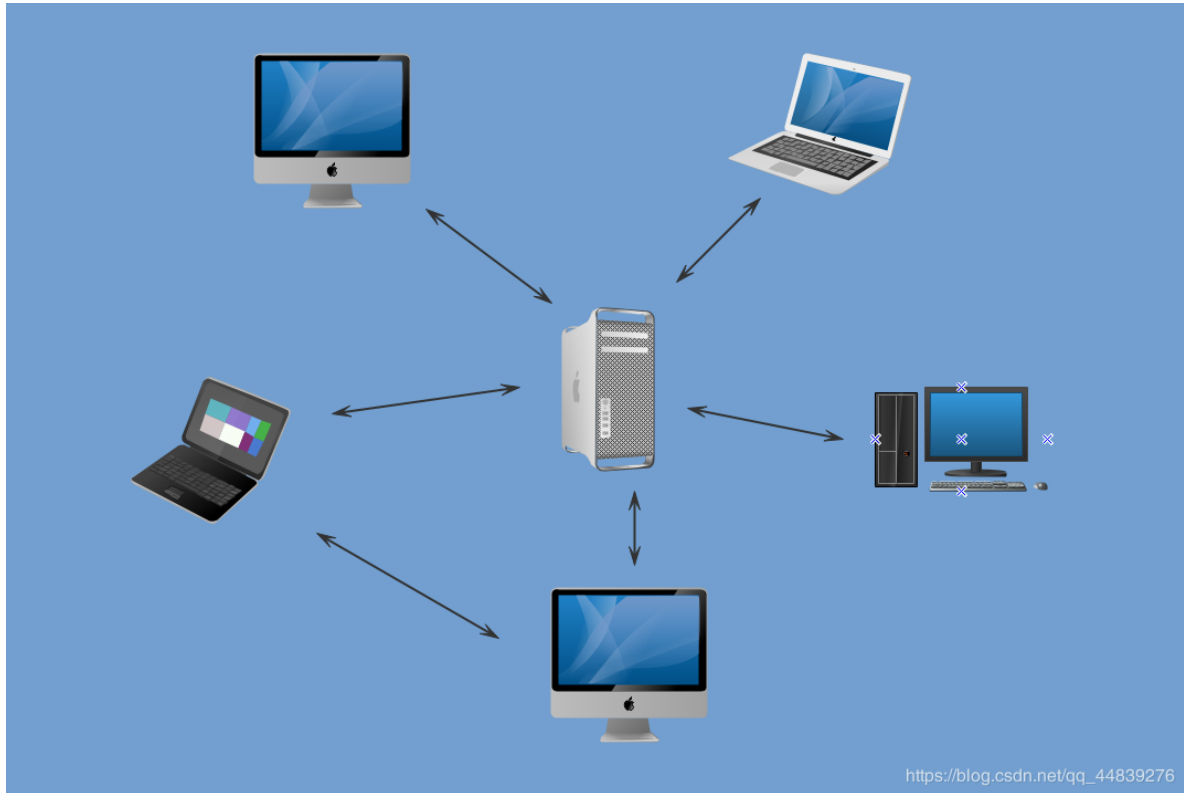
分布式版本控制系统有Git

**集中式版本控制系统：**你的版本库存放在中央服务器上，而且必须得联网，首先你得用自己的电脑拉取代码到本地，然后在进行修改，改完了之后再上传，这是你自己一个人的操作，如果是很多人或是在一个局域网内进行操作，那么你的网速将会变得很慢很慢，传文件也会非常慢，很影响心情，这样你就得加班了，你说是不是。还有，如果你的中央服务器挂了之后，那么所有的工作都白做了，这一个月你都得上加班。。。。。



**分布式版本控制系统：**首先分布式版本系统没有所谓的中央服务器，每个人的电脑就是一个版本库，那么如何保障多人同时协作呢，你改了A文件，同事也改了A文件，这时候只需要你把自己修改的东西发给你的同事就行了。但是两个人也不能就这样传来传去啊是不，其实在分布式版本控制系统当中，也有一台服务器来充当“中央服务器”，只不过和集中式的区别是这台中央服务器挂了之后，你还能干活，不会像集中式那样，一旦挂了之后全员下班的样子。分布式这台中央服务器的作用就是充当了一个中间商的作用，用来交换你们彼此的文件，有没有这个中间商都能干活，只不过是方便与否罢了。最后

他还有强大的分支管理功能，后面详谈。



## 四、安装git，创建版本，git命令的使用

### 1. linux下yum源安装

```
yum -y install git
```

### 2. 安装后进行设置，方便每个人进行识别

```
--global全局配置，在你机器上所有的git仓库都会使用这个信息  
git config --global user.name "zhanghaodong"  
git config --global user.email "zhanghaodong@kylinos.cn"
```

### 3. 创建版本库，用git init这个命令把mygit目录变成可管理的仓库，ls -a可见

```
mkdir gitfile && cd gitfile && git init  
[root@pc01 gitfile]# ls -a  
.  ..  .git  
#.git的结构  
[root@pc01 gitfile]# tree .git  
.git  
├── branches  
├── config  
├── description  
├── HEAD  
├── hooks  
│   ├── applypatch-msg.sample  
│   ├── commit-msg.sample  
│   ├── post-update.sample  
│   ├── pre-applypatch.sample  
│   ├── pre-commit.sample  
│   └── prepare-commit-msg.sample
```

```
|   ├── pre-push.sample
|   ├── pre-rebase.sample
|   └── update.sample
└── info
    └── exclude
objects
|   ├── info
|   └── pack
└── refs
    ├── heads
    └── tags
```

9 directories, 13 files

#### 4.检查仓库有无添加/提交内容

```
#可以看到工作区是干净的
[root@pc01 gitfile]# git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

#### 5.创建文件并添加和提交

##### (1) 第一次写文件

```
#编辑文件
[root@pc01 gitfile]# vim README.txt
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
#检查工作区情况
[root@pc01 gitfile]# git status
# 位于分支 master
#
# 初始提交
#
# 未跟踪的文件:
#   (使用 "git add <file>..." 以包含要提交的内容)
#
#   README.txt
提交为空, 但是存在尚未跟踪的文件 (使用 "git add" 建立跟踪)
#使用add命令提交到暂存区, 然后用commit在提交到版本库就可
[root@pc01 gitfile]# git add README.txt
[root@pc01 gitfile]# git status
# 位于分支 master
#
# 初始提交
#
# 要提交的变更:
#   (使用 "git rm --cached <file>..." 撤出暂存区)
#
#   新文件:       README.txt
#
[root@pc01 gitfile]# git commit -m "the first commit"
[master (根提交) 36d833c] the first commit
```

```
1 file changed, 1 insertion(+)
create mode 100644 README.txt
#再次检查工作区情况, 这回世界干净了。
[root@pc01 gitfile]# git status
# 位于分支 master
无文件要提交, 干净的工作区
```

(2) 第二次写文件, 添加The second write for gitfile!!!这一行内容, 然后进行提交, 使用git log查看版本历史

```
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
The second write for gitfile!!!
[root@pc01 gitfile]# git log
commit 40c79f4f678847e6cff917ae7d682f5fd869ad0b
Author: zhanghaodong <zhanghaodong@kylinos.cn>
Date: Thu Jul 23 10:46:29 2020 +0800

    the second commit

commit 36d833ce6360e0c63cd0ade97545ce07a5b6bc9e
Author: zhanghaodong <zhanghaodong@kylinos.cn>
Date: Thu Jul 23 10:38:02 2020 +0800

    the first commit
#现在查看README.txt内容
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
The second write for gitfile!!!
```

(3) 现在后悔了, 想要第一次修改的版本, 怎么办, 么着急, 使用git reset --hard commit\_id进行版本回退就行了

```
#commit_id是你提交的版本id
[root@pc01 gitfile]# git reset --hard 36d8
HEAD is now at 36d833c the first commit
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
```

(4) 当我写文件时候, 在没add或commit之前, 把我工作区的修改给清空怎么办, 别着急, Don't worry, 用这个git checkout -- filename,下面演示一下

```
#首先检查一下工作区, 干净的
[root@pc01 gitfile]# git status
# On branch master
nothing to commit, working directory clean
#写文件测试
[root@pc01 gitfile]# echo "this third write gitfile" >> README.txt
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
this third write gitfile
#检查工作区有待提交的
[root@pc01 gitfile]# git status
# On branch master
```

```

# 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)
#
#   modified:   README.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
#现在我想把工作区的内容撤销
[root@pc01 gitfile]# git checkout -- README.txt
#再次检查工作区发现是干净的, 已经撤销完毕了
[root@pc01 gitfile]# git status
# On branch master
nothing to commit, working directory clean
#检查文件还是最初始的状态
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!

```

(5) 现在你在文件中写了一行这样的话The stupid boss doesn't know what to do, 由于你的胆小你想撤回了, 你已经提交到暂存区了, 现在离提交只有一步之遥了, 这个时候怎么办, 在线等, 挺急的。行吧, 看下面的大招。

```

[root@pc01 gitfile]# echo "The stupid boss doesn't know what to do" >>
README.txt
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!
The stupid boss doesn't know what to do
[root@pc01 gitfile]# git add .
[root@pc01 gitfile]# git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README.txt
#

#git reset回到当前最新的版本, 就是你add之前的版本
[root@pc01 gitfile]# git reset HEAD README.txt
Unstaged changes after reset:
M   README.txt
#查看工作区, 果然文件被修改了
[root@pc01 gitfile]# git status
# On branch master
# 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)
#
#   modified:   README.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
#现在撤回对工作区的修改, 老板当做我什么都没有发生。
[root@pc01 gitfile]# git checkout -- README.txt
[root@pc01 gitfile]# git status
# On branch master
nothing to commit, working directory clean
[root@pc01 gitfile]# cat README.txt
Git is friendly to me!!!

```

---

小结：

**A.工作区和暂存区**

git分为工作区、暂存区。当前目录所在的目录就是工作区，在工作区修改了代码后，要add提交到暂存区，然后在commit提交到版本库中。

**B.管理和修改**

修改了工作区的代码，还没有add时，可用git checkout -- filename进行撤销，这条命令的作用就是撤销对工作区代码的修改

**C.管理修改**

修改了工作区的代码，已经add到了暂存区，但是这个时候想撤销怎么办，用这两条命令。首先git reset HEAD filename（意思是默认回到了当前最新的版本，也就是在你add之前的版本），然后再用git checkout -- filename撤销对工作区的修改，这样整个世界就清净了。

**D.版本回退**

回退版本，命令是git reset --hard commit\_id（这里commit\_id可以通过git log查看）

**E.git checkout**

这个命令作用就是，当你的代码已经提交到版本库了，这个时候你把工作区的所有文件都删除了，不要着急，用git checkout -- filename就可以进行轻松的恢复了。

## git命令使用

序号	命令	说明
1	git add .	提交工作区的内容到暂存区
2	git commit -m "your comment"	将暂存区的内容提交到版本库
3	git checkout -- filename	撤销对工作区代码的修改
4	git reset HEAD filename	回退到最新版本/撤销暂存区的代码
5	git reset --hard commit_id	回退到某一历史版本
6	git branch	显示当前所在分支（也可以查看所有分支）
7	git branch 分支名称	创建分支
8	git branch -b 分支名称 或 git switch -c 分支名称	创建并切换分支
9	git branch -d 分支名称	删除分支
10	git checkout 分支名称 或 git switch 分支名称	切换分支
11	git merge 分支名称	合并分支
12	git clone	克隆复制
13	git pull	拉取
14	git push	上传
15	git log --pretty=oneline --graph --abbrev-commit	清楚显示版本信息(简洁还有图谱呢)
16	git remote -v	显示本地远程仓库
17	git remote rm 远程仓库名字（自己取得）	删除本地远程仓库
18	git remote add 名字 地址	添加远程仓库
19	git stash	隐藏当前工作状态
20	git stash list	查看已隐藏的内容
21	git apply stash@{x}	恢复到x工作状态，但不删除stash内容
22	git stash drop	删除stash内容
23	git stash pop	恢复到工作状态并删除stash
24	git cherry-pick commit_id	复制提交bug的commit_id到当前分支
25	git rebase branch_name	简化提交历史



## 五、创建分支、合并分支（小试）

- (1) 创建分支dev，切换分支dev，编写代码  
(分支前面的星号代表当前所在分支)

```
[root@pc01 ~]/gitfile]#git branch dev #创建dev分支
[root@pc01 ~]/gitfile]#git checkout dev #切换dev分支
Switched to branch 'dev'
[root@pc01 ~]/gitfile]#git branch #查看当前所在分支
* dev
  master
#查看版本历史，新建的分支会默认含有主分支的log信息，下面这四个是master分支的log信息
[root@pc01 ~]/gitfile]#git log --pretty=oneline
69e92f8cd1859e733d611b6975139fd224afdd50 delete my_dev
f7af52ef156223b67f1815f3f400325fd155f9f3 add test comment
26e7a8d6a97654f5d427572edc138cebb2efbe0c add test to my_dev
36d833ce6360e0c63cd0ade97545ce07a5b6bc9e the first commit
#编写代码，并提交，这样一个分支建立好了
[root@pc01 ~]/gitfile]#echo "I'm create new branch,her name is dev!!" >
mydev.sh
echo "I'm create new branch,her name is devgit branch" > mydev.sh
[root@pc01 ~]/gitfile]#git add .
[root@pc01 ~]/gitfile]#git commit -m "create my_dev.sh test"
[dev 854e018] create my_dev.sh test
 1 file changed, 1 insertion(+)
  create mode 100644 mydev.sh
[root@pc01 ~]/gitfile]#git log --pretty=oneline
854e01876e5dc1b845bcec9074fae6c4e9d72bf0 create my_dev.sh test
69e92f8cd1859e733d611b6975139fd224afdd50 delete my_dev
f7af52ef156223b67f1815f3f400325fd155f9f3 add test comment
26e7a8d6a97654f5d427572edc138cebb2efbe0c add test to my_dev
36d833ce6360e0c63cd0ade97545ce07a5b6bc9e the first commit
```

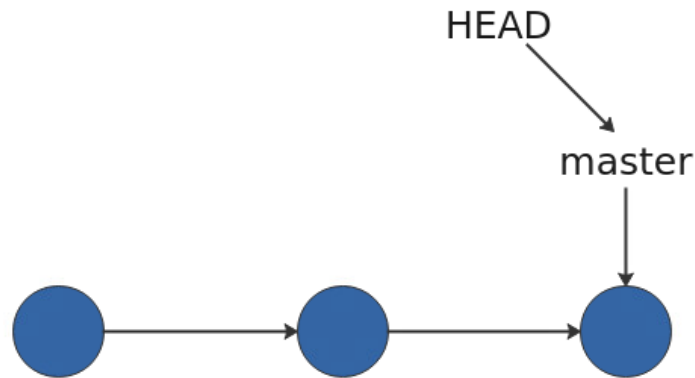
- (2) 合并分支

```
#查看当前分支
[root@pc01 ~]/gitfile]#git branch
* dev
  master
#切换到主分支进行合并
#为什么要切换到主分支进行合并呢，因为在master分支上没有dev这个分支的信息，但是你新建的dev的分支，默认就会有master分支的log信息和版本信息，所以合并的时候就会提示Already up-to-date.这个信息了（错误理解，稍后改正）。
[root@pc01 ~]/gitfile]#git checkout master
Switched to branch 'master'
[root@pc01 ~]/gitfile]#git merge dev
Updating 69e92f8..854e018
Fast-forward
 mydev.sh | 1 +
 1 file changed, 1 insertion(+)
  create mode 100644 mydev.sh
[root@pc01 ~]/gitfile]#ls
mydev.sh  README.txt
[root@pc01 ~]/gitfile]#cat mydev.sh
```

```
I'm create new branch,her name is devgit branch
```

## 六、图文展示创建分支与合并分支原理

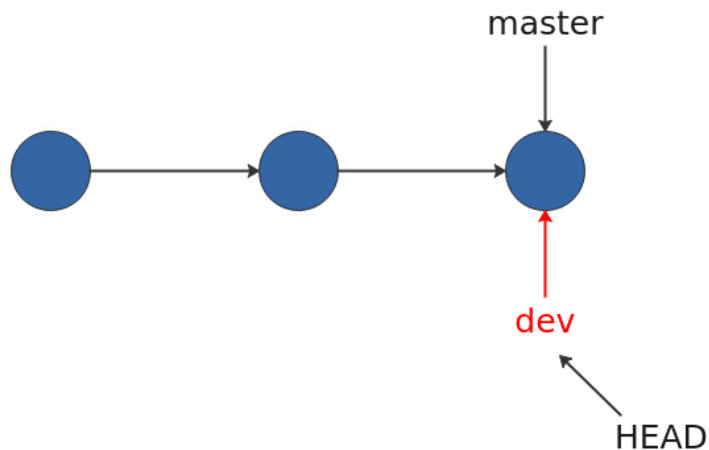
在你git安装好，你修改完代码的时候，默认就会有一个分支名叫master，一般叫主分支。HEAD指针严格说不是指向的提交，而是指向的master，master才是指向提交的，所以HEAD指的是当前的分支。



[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

在你每次提交的时候，master分支就会向前移动一步，越来越长越来越长。

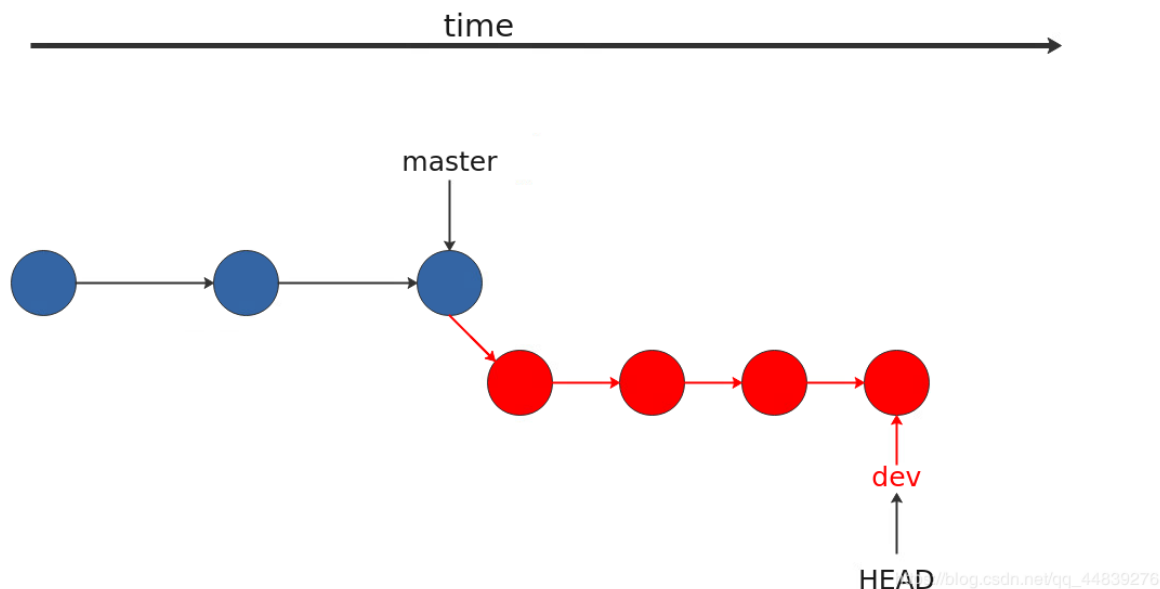
当创建新的分支时，假如我新创建的分支叫dev，那么git就会新建一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前我们在dev分支上。



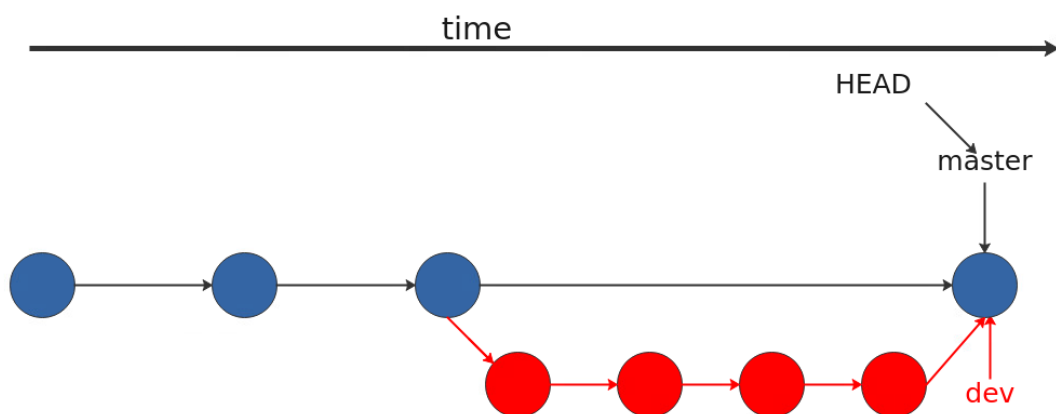
[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

所以Git创建分支很快，因为增加一个dev指针，改变了HEAD的指向，工作区的文件没有任何的变化。所以现在开始，我们修改代码什么的操作就是在dev这个分支上进行了，如下图所示，红色的圈圈表示我们修改了几次代码，这几次的代码的修改我们都是在dev这个分支上进行操作的，提交一次代码

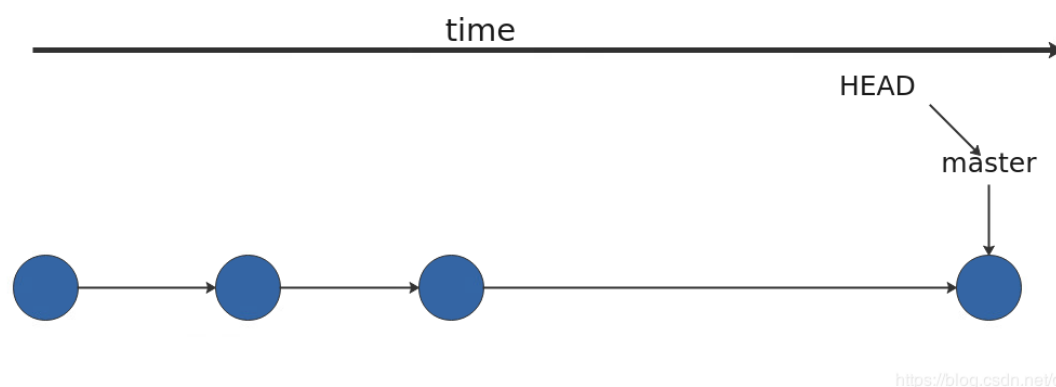
后，dev指针就会往前移动一步，而master不变。



不一会，我们代码写完现在要把dev合并到master上，怎么合并呢，很简单直接把master的指针指向dev当前的提交就ok了（git merge master）。



所以嘛，合并也是很快的，就是把master的指针指向dev的当前提交就ok了，代码也不用变。合并完以后，你感觉没用了，当然也可以把dev分支删除掉，最后就剩下master自己了，是不是有点可怜了。



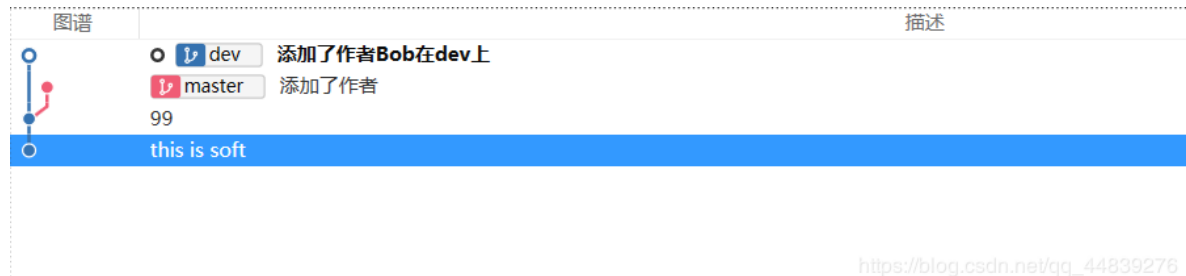
## 七、分支管理（综合）

### （一）、创建与合并分支

默认分支名字叫做master，随着每一次的提交，master的这个分支会越来越长。

```
#新版本git现在也支持switch进行切换了，避免和checkout搞混
git branch dev #创建
git checkout dev = git switch dev#切换
git branch -b dev = git switch -c dev #创建并切换
```

## (二)、合并冲突怎么办



如上图所示一样，现在在master分支上README内容如下：

```
README
块 1：行 1-4
1 1 ...this is software
2 2 ...this is 8888 software
3 3 ...this is SOFTWARE
4 4 + i am natasha
```

而dev上README内容如下：

```
README
块 1：行 1-4
1 1 ...this is software
2 2 ...this is 8888 software
3 3 ...this is SOFTWARE
4 4 + i am bob
```

然后现在进行合并，报错如下（其实也不是报错，只不过是有了冲突）：

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git merge master
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ git status
On branch dev
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

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

https://blog.csdn.net/qq\_44839276

其实这只不过是冲突了，git不知道到底要保留哪一个，这时候需要我们手动进行选择，这个时候查看README内容如下：

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ cat README
this is software
this is 8888 software
this is SOFTWARE
<<<<<<< HEAD
i am bob
=====
i am natasha
>>>>>>> master
```

这里git用

<<<<<<<HEAD      =====      >>>>>>>>>master"

来标记处不同分支的内容。在这里我修改成了这个样子“i am bob and natasha”

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ vim README

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ cat README
this is software
this is 8888 software
this is SOFTWARE
i am bob and natasha

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ git add .

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev|MERGING)
$ git commit -m '解决了冲突'
[dev 7c66c73] 解决了冲突

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git log --pretty=oneline --graph
* 7c66c737f040af7ae1cbf7aa87e3e949ebafe5fa (HEAD -> dev) 解决了冲突
|
| * 42cee6a5108fea588028b8b5aa9cc1d032f7ef0f (master) 添加了作者
| * a48c4963bea942f91baa58f256d9fba7d588114b 添加了作者Bob在dev上
|/
* bb9c62e82e01924600e2e681388800c5dc4c3c5f 99
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft
```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

再看清晰的看一下分支图谱：

The screenshot shows the Git GUI interface. On the left, the '图谱' (Graph) tab is selected, showing a commit graph with 'dev' and 'master' branches. The 'dev' branch is highlighted with a red box. On the right, the '提交' (Commits) table lists several commits, with the top one being '7c66c73' by 'RANRAN-PC\Jingxiang' with the message '解决了冲突'. Below the table, the 'README' file diff is shown, with a red box highlighting the changes: '- i am bob' and '+ i am bob and natasha'.

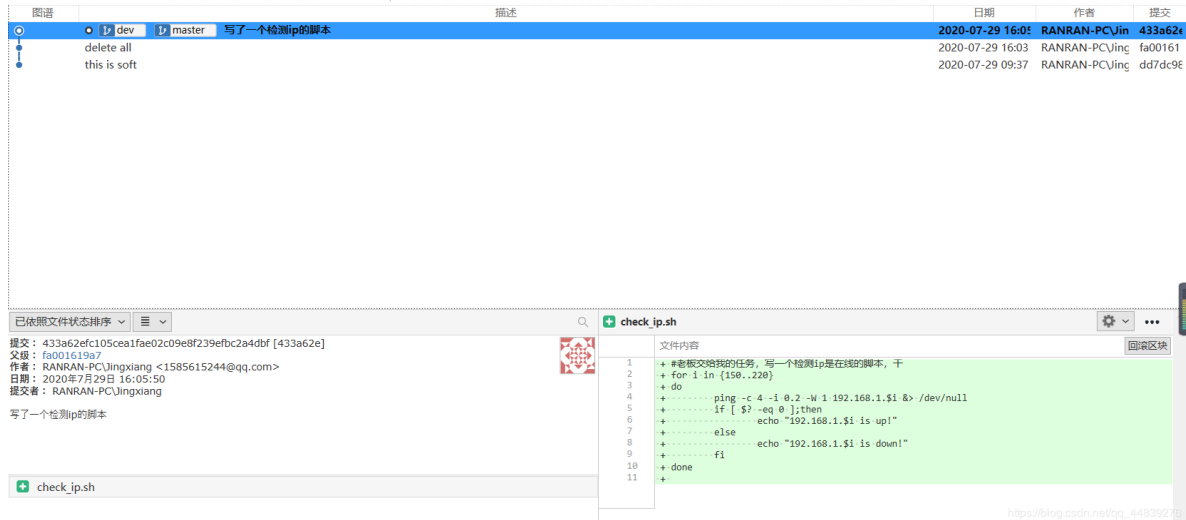
总结：

- (1) 在合并的时候，有时候会显示Fast Forward（快速合并），但有时候就出现这样的conflict test（冲突文件）了，这时手动确认解决最安全。
- (2) 当git无法完成自动合并分支时，这时候需要我们手动进行解决，解决冲突后，再提交，再合并。（解决冲突的意思就是手动把git合并失败的文件搜手动修改成我们希望的内容后，在进行提交）

### （三）、合并冲突在什么时候会出现

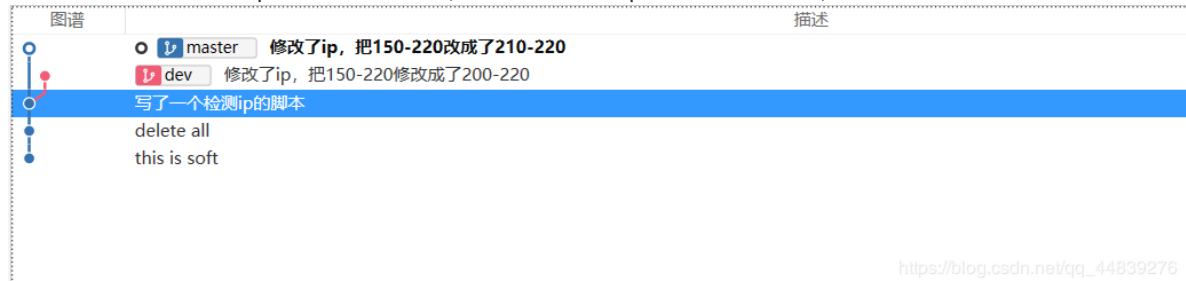
我们在合并的时候，有时候git会使用Fast Forward模式快速合并，而有的时候会出现conflict text，那么什么时候会出现这两种情况呢？？你来看：

(1) 两个分支有相同文件的时候，修改同一地方



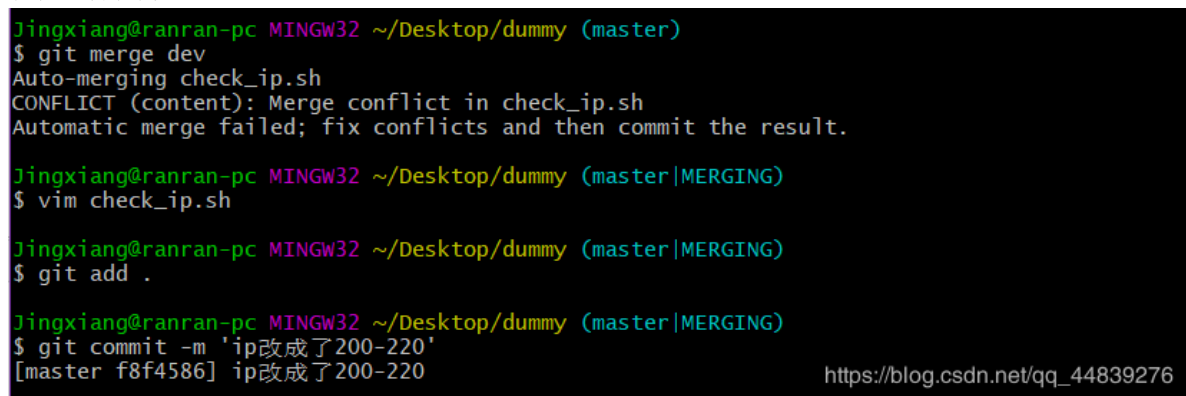
The screenshot shows the Git GUI interface. At the top, there's a commit history table with columns: 图谱, 描述, 日期, 作者, 提交. The first commit is highlighted: 433a62e, dated 2020-07-29 16:03, by RANRAN-PC\Jing, with the description '写了一个检测ip的脚本'. Below the history, the file 'check\_ip.sh' is selected, showing its content in a diff view. The content is a shell script for checking IP connectivity. The bottom status bar shows '已依照文件状态排序'.

现在master分支修改ip范围是210-220，dev分支修改ip范围是200-220，修改后如下：



The screenshot shows the Git GUI interface. The commit history table has columns: 图谱, 描述. The first commit is highlighted: 433a62e, dated 2020-07-29 16:03, by RANRAN-PC\Jing, with the description '写了一个检测ip的脚本'. Below the history, the file 'check\_ip.sh' is selected, showing its content in a diff view. The bottom status bar shows '已依照文件状态排序'.

现在进行合并：



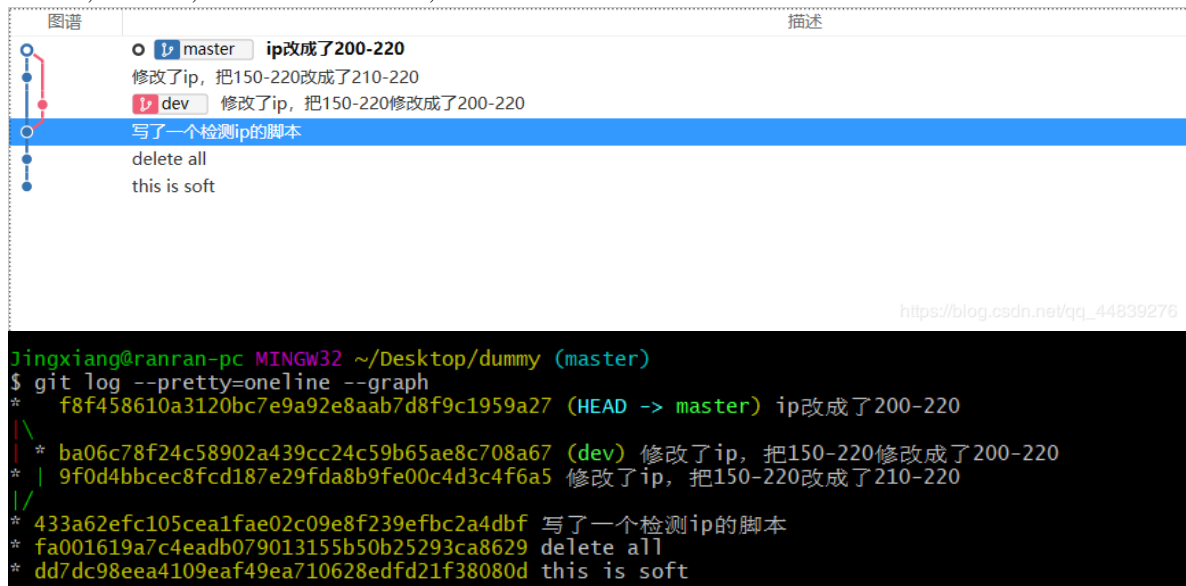
```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev
Auto-merging check_ip.sh
CONFLICT (content): Merge conflict in check_ip.sh
Automatic merge failed; fix conflicts and then commit the result.

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master|MERGING)
$ vim check_ip.sh

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master|MERGING)
$ git add .

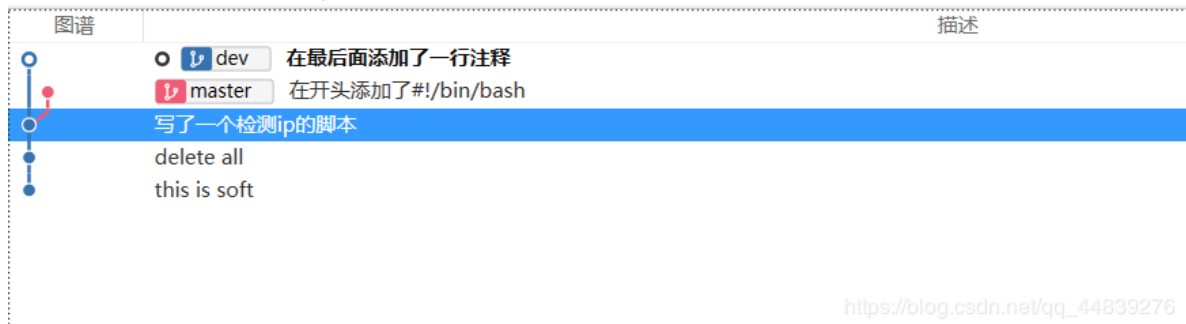
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master|MERGING)
$ git commit -m 'ip改成了200-220'
[master f8f4586] ip改成了200-220
```

很显然，冲突了，手动解决后提交ok，最后是这样的



The screenshot shows the Git GUI interface. The commit history table has columns: 图谱, 描述. The first commit is highlighted: 433a62e, dated 2020-07-29 16:03, by RANRAN-PC\Jing, with the description '写了一个检测ip的脚本'. Below the history, the file 'check\_ip.sh' is selected, showing its content in a diff view. The bottom status bar shows '已依照文件状态排序'.

(2) 两个分支有相同文件的时候，修改不同地方  
分别在开头和末尾添加代码，然后进行合并



在合并的时候，会跳出一个文本文件，意思大概就是让你写一个注释，写就完了。

合并后见下：

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev
Auto-merging check_ip.sh
Merge made by the 'recursive' strategy.
  check_ip.sh | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
*   d54012278d21924ddc9462d64a2ac2e9a0ea45f (HEAD -> master) 合并了master和dev
| \
| * 231fea7fa673d9faa81eb9b52f1142d608fc8285 (dev) 在最后面添加了一行注释
| | b344ff6862fefce0c3d86a01aeb1fd3066b443c3 在开头添加了#!/bin/bash
|/
* 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
* fa001619a7c4eadb079013155b50b25293ca8629 delete all
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ cat check_ip.sh
#!/bin/bash
#老板交给我的任务，写一个检测ip是在线的脚本，干
for i in {150..220}
do
    ping -c 4 -i 0.2 -w 1 192.168.1.$i &> /dev/null
    if [ $? -eq 0 ];then
        echo "192.168.1.$i is up!"
    else
        echo "192.168.1.$i is down!"
    fi
done
#改脚本仅供参考
```

→ master分支改的

→ dev分支改的

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

也可以用这种方式进行合并，禁用Fast Forward模式，使用--no-ff参数

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev --no-ff -m '合并dev分支文件'
Auto-merging check_ip.sh
Merge made by the 'recursive' strategy.
  check_ip.sh | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ cat check_ip.sh
#!/bin/bash
#老板交给我的任务，写一个检测ip是在线的脚本，干
for i in {150..220}
do
    ping -c 4 -i 0.2 -w 1 192.168.1.$i &> /dev/null
    if [ $? -eq 0 ];then
        echo "192.168.1.$i is up!"
    else
        echo "192.168.1.$i is down!"
    fi
done
#改脚本仅供参考

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
*   e326cd35926ca16a578e128f499a36e44a33fa70 (HEAD -> master) 合并dev分支文件
| \
| * 231fea7fa673d9faa81eb9b52f1142d608fc8285 (dev) 在最后面添加了一行注释
| | b344ff6862fefce0c3d86a01aeb1fd3066b443c3 在开头添加了#!/bin/bash
|/
* 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
* fa001619a7c4eadb079013155b50b25293ca8629 delete all
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft
```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

最后图形化合并后：



(3) 两个分支有不同文件的时候

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ ls
check_ip.sh  test.sh*

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git switch master
Switched to branch 'master'

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ ls
check_ip.sh*  README
```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

合并的时候会提示你添加注释，如果你不想添加的话，敲一下回车就行了（默认的注释就是Merge branch 'dev'）

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev
Merge made by the 'recursive' strategy.
 test.sh | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test.sh

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ ls
check_ip.sh*  README  test.sh*

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git status
On branch master
nothing to commit, working tree clean

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
* a207eaf76591185fbf93be04e933a1f7af6dd183 (HEAD -> master) Merge branch 'dev'
|
| * 4cfa43e087d1289038521d5afa309797ed8a42c8 (dev) 添加了一个test.sh脚本
| * e340f9204948f3d4a974632e1a91d77ec072615d 添加了i am well
| * e326cd35926ca16a578e128f499a36e44a33fa70 合并dev分支文件
|
| * 231fea7fa673d9faa81eb9b52f1142d608fc8285 在最后面添加了一行注释
| * b344ff6862fefce0c3d86a01aeb1fd3066b443c3 在开头添加了#!/bin/bash
|
| * 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
| * fa001619a7c4eadb079013155b50b25293ca8629 delete all
| * dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft
```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)



(4) 一般master分支是雷打不动的（且你是不能再这上面干活的），把其他分支合并到此分支上（团队开发常用，建议加上--no-ff，后期查看版本历史）。

查看版本历史的时候，没有加--no-ff，就看不到什么时候进行合并的，相反加上就会很清楚的看到。第



一张是没有用参数，第二张用了参数。

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git switch master
Switched to branch 'master'

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev
Updating 433a62e..df01ef4
Fast-forward
 check_ip.sh | 1 +
 1 file changed, 1 insertion(+)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ cat check_ip.sh
#老板交给我的任务，写一个检测ip是在线的脚本，干
#这个脚本修改完了要提交到master上
for i in {150..220}
do
    ping -c 4 -i 0.2 -w 1 192.168.1.$i &> /dev/null
    if [ $? -eq 0 ];then
        echo "192.168.1.$i is up!"
    else
        echo "192.168.1.$i is down!"
    fi
done

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
* df01ef45ef671d6e58ed0240242c253cedece644 (HEAD -> master, dev) 添加了第二行注释
* 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
* fa001619a7c4eadb079013155b50b25293ca8629 delete all
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft https://blog.csdn.net/qq_44839276

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev --no-ff -m '合并dev到master'
Merge made by the 'recursive' strategy.
 check_ip.sh | 1 +
 1 file changed, 1 insertion(+)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
* e23c0c575c44d24c91a20e8647443fc35ede4472 (HEAD -> master) 合并dev到master
* df01ef45ef671d6e58ed0240242c253cedece644 (dev) 添加了第二行注释
* 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
* fa001619a7c4eadb079013155b50b25293ca8629 delete all
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft https://blog.csdn.net/qq_44839276
```

## 总结

- (1) 在修改相同文件的同一处的时候，提交合并的时候会有冲突，手动解决即可。
- (2) 修改不同文件的时候，进行合并，合并后跳出添加注释的界面，想添加就添加，不想添加就回车。

## (四)、分支管理策略

```

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
* 1ce5f3a5d0a58435f7d74b3c050f4e86b5e8da13 (HEAD -> master, dev) 添加了no-ff模式
* 7c66c737f040af7ae1cbf7aa87e3e949ebafe5fa 解决了冲突
|
| * 42cee6a5108fea588028b8b5aa9cc1d032f7ef0f 添加了作者
| * a48c4963bea942f91baa58f256d9fba7d588114b 添加了作者Bob在dev上
|/
* bb9c62e82e01924600e2e681388800c5dc4c3c5f 99
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git reset --hard 7c66
HEAD is now at 7c66c73 解决了冲突

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git merge dev --no-ff -m 'merge without no-ff'
Merge made by the 'recursive' strategy.
 README | 1 +
 1 file changed, 1 insertion(+)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (master)
$ git log --pretty=oneline --graph
* 5dd83fe6210199e6acd9439a5d4994304771233a (HEAD -> master) merge without no-ff
|
| * 1ce5f3a5d0a58435f7d74b3c050f4e86b5e8da13 (dev) 添加了no-ff模式
| * 7c66c737f040af7ae1cbf7aa87e3e949ebafe5fa 解决了冲突
|
| * 42cee6a5108fea588028b8b5aa9cc1d032f7ef0f 添加了作者
| * a48c4963bea942f91baa58f256d9fba7d588114b 添加了作者Bob在dev上
|/
* bb9c62e82e01924600e2e681388800c5dc4c3c5f 99
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft

```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

在git进行合并分支的时候，如果可能，git会使用Fast Forward模式进行快速合并，但是这个时候，合并了分支以后，把该分支删除掉，就会丢失该分支的信息了。

如果要强制禁用掉Fast Forward模式后，git在merge的时候就会生成一个新的commit，这样我们就可以看出这个分支的历史信息。

禁用模式使用的命令是**--no-ff**。

上图中第一个git log，我没有使用--no-ff参数进行合并，可以看出graph不是很明显。下面的git log显示的是我用了--no-ff之后，可以看出在合并的时候显示dev的历史信息。（这样经过了很长时间过去之后，再回来看依然知道这一块是合并了什么）

### 分支策略

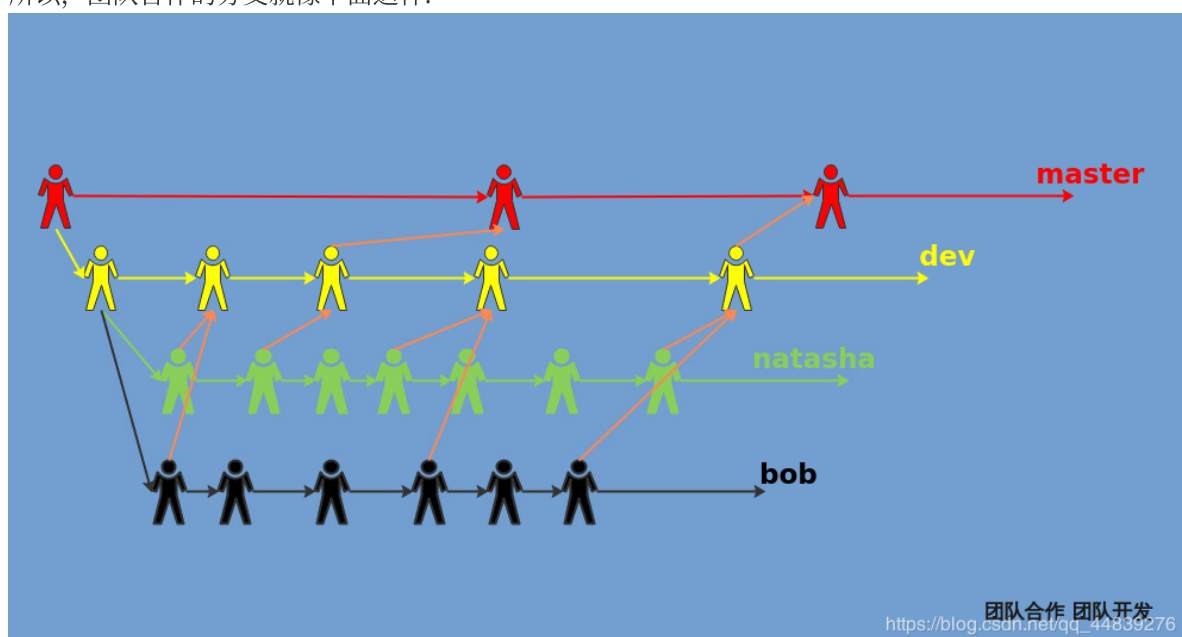
在实际的工作环境下，我们应该按照几个基本原则来进行分支的管理。

(1) master分支之所以叫master分支，是因为他是老大，这个分支是非常稳定的，我们平常的工作不能在这上面完成，这上面只能够用来发布新版本，ok??? yes!!!

(2) 那在哪干活工作呢，在dev分支上，这个分支是最不稳定的，你在dev上开发1.0的开发完事了，现在需要合并到master上，用master分支来正式发布新的1.0版本

(3) 现在和你的小伙伴在dev分支上干活吧，每个人在dev分支上又有自己的分支，写完了代码时不时的合并到dev上就行了。

所以，团队合作的分支就像下面这样：



### 总结

在合并分支时候，加上--no-ff参数实现普通模式合并，合并后的历史有分支，能看出来曾经做过合并。

（Fast Forward看出来哦！）

## （五）、Bug分支之隐藏当前工作区

当我们在开发的过程中，总会遇到bug（无处不在的bug，当然码农的工作就是解决bug了），怎么解决？新建一个bug分支，修改代码，合并，删除bug分支，ok。

这个时候经理(foolish)过来告诉你一个bug，说你的ip检测上限变成了230，要把220修改成230，这个时候你就想创建一个bug分支用来解决这一bug，bug分支的名字叫rewrite\_ip，但是这个时候你手头还有工作，你还有没有完成的代码，而且还没有提交。不是你不提交，而是完成这个任务需要大概一天的时间，而解决这个bug只需要你几分钟的时间就完事了，怎么办，当前还有工作没有提交：

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git status
On branch dev
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:   check_ip.sh

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

还好，git有一个躲猫猫的功能（隐藏），就是把你当前的工作状态给隐藏起来，命令就是stash：

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git stash
Saved working directory and index state WIP on dev: df01ef4 添加了第二行注释

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git status
On branch dev
nothing to commit, working tree clean
```

现在工作区干净了，现在新建bug分支开始修复吧

```

MINGW32:/c/Users/Jingxiang/Desktop/dummy
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git switch -c rewrite_ip
Switched to a new branch 'rewrite_ip'

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (rewrite_ip)
$ vim check_ip.sh

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (rewrite_ip)
$ cat check_ip.sh
#老板交给我的任务，写一个检测ip是在线的脚本，干
#这个脚本修改完了要提交到master上
for i in {150..230}
do
    ping -c 4 -i 0.2 -W 1 192.168.1.$i &> /dev/null
    if [ $? -eq 0 ];then
        echo "192.168.1.$i is up!"
    else
        echo "192.168.1.$i is down!"
    fi
done

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (rewrite_ip)
$ git add .

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (rewrite_ip)
$ git commit -m 'change ip 220 for 230'
[rewrite_ip e9d5d2a] change ip 220 for 230
1 file changed, 1 insertion(+), 1 deletion(-)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (rewrite_ip)
$ git switch dev
Switched to branch 'dev'

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git merge --no-ff -m '修复了bug, 改了ip' rewrite_ip
Merge made by the 'recursive' strategy.
check_ip.sh | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git log --pretty=oneline --graph
* 528134b400885f8eed1e07fd376be9abef51336c (HEAD -> dev) 修复了bug, 改了ip
| \
| * e9d5d2afdc1f1681309790b7ed80400d0ff50645 (rewrite_ip) change ip 220 for 230
| /
* df01ef45ef671d6e58ed0240242c253cedece644 添加了第二行注释
* 433a62efc105cea1fae02c09e8f239efbc2a4dbf 写了一个检测ip的脚本
* fa001619a7c4eadb079013155b50b25293ca8629 delete all
* dd7dc98eea4109eaf49ea710628edfd21f38080d this is soft

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git status
On branch dev
nothing to commit, working tree clean

```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

那么现在bug修复完了，以前的做的工作哪去了，别着急，用git stash list查看

```

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git stash list
stash@{0}: WIP on dev: df01ef4 添加了第二行注释

```

那怎么恢复呢，两种方法

- (1) git stash apply，但是恢复后stash内容没有删除，你需要再用git stash drop进行删除
- (2) git stash pop，恢复的同时把stash的内容也删除了

当有多行的stash时，用git stash apply stash@{}进行恢复

```

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git stash list
stash@{0}: WIP on dev: 2f5e6e9 stash
stash@{1}: WIP on dev: 2f5e6e9 stash
stash@{2}: WIP on dev: df01ef4 添加了第二行注释

Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (dev)
$ git stash apply stash@{2}
On branch dev
nothing to commit, working tree clean

```

假设另一种情况，在master修复了bug后，因为早期的dev分支是从master上进行分出来的，所以dev分支也存在这个问题，那么我们在dev上在进行一次修复bug的操作吗，可以，只不过是复杂了。git提供了一个命令叫cherry-pick，作用是我们可以复制一个特定的提交到当前的分支，这样当前分支的bug就解决了。

切换到要修复的分支，执行git cherry-pick commit\_id进行复制（这个commit\_id指的是你在bug分支

上进行提交后的commit\_id/提交bug的commit\_id），这样git就会自动给你当前所在的分支进行提交。

总结：

- (1) 出现bug时，新建bug分支，然后修复-合并-删除
- (2) 当你手头还有没有完成的工作，先躲猫猫stash一下，然后去修复bug，修复后，返回来用git stash pop再回到工作现场就ok了。
- (3) 在dev上修复的bug，想要合并到master分支，用git cherry-pick commit\_id（commit\_id指的是你提交的bug的commit\_id），把bug提交的修改复制到当前分支就行。

## （六）、Feature分支

开发一个新的功能，新建一个分支，完事之后删除，删除不了要强制删除，使用选项-D

```
git branch -D name
```

## （七）、多人协作

当你从远程仓库进行clone时，git自动把本地的master和远程的master关联了起来，并且远程仓库的默认名称是origin。

查看远程仓库的信息，如果没有推送的权限就看不到push的地址

```
git remote -v
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (1-branch)
$ git remote -v
remotely      https://github.com/zhd-haodong/0730study.git (fetch)
remotely      https://github.com/zhd-haodong/0730study.git (push)
```

推送分支

推送分支就是把所有本地提交推送到远程仓库上，推送的时候要指定本地分支，这样就会把分支推送到远程仓库上对应的远程分支。

```
git push remotely master #推送本地分支master到远程库
git push remotely dev    #推送本地分支dev到远程库
```

## （八）、Rebase

将多岔路的提交历史变成直线

```
git rebase branch_name
```

# 八、标签管理

## （一）、创建标签

打标签非常简单，他打的是当前已经提交的内容。也可以打过去提交的commit\_id。

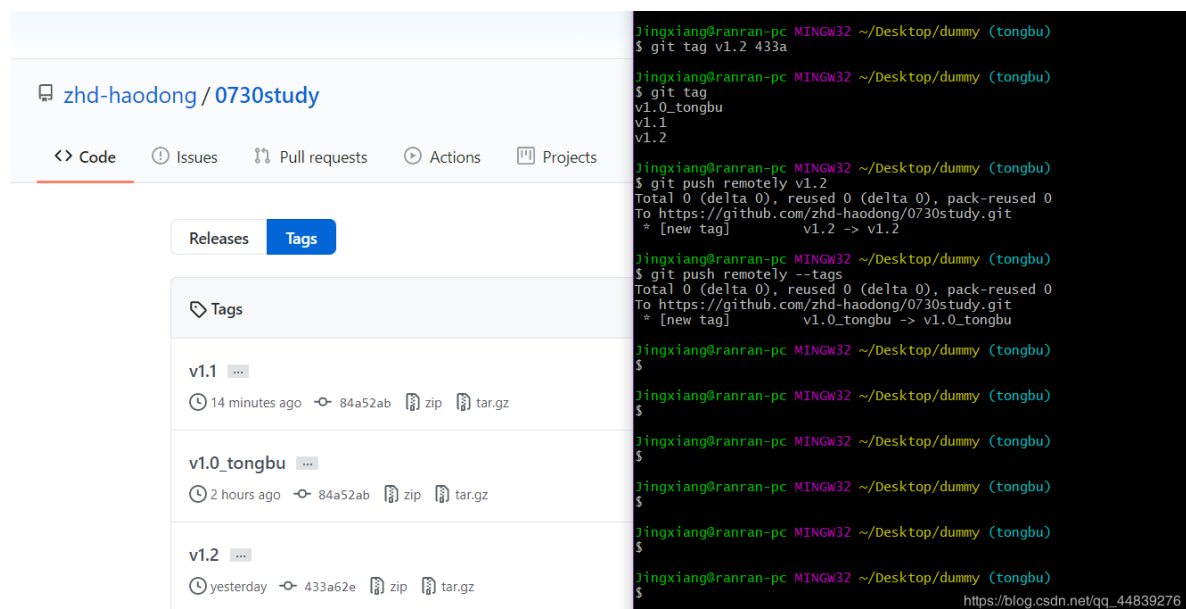
```
git tag <name> #默认是HEAD, 也可以commit_id
git tag v1.0 #打标签, 名字是v1.0
git tag #查看所有标签
git show v1.0 #查看详细信息
git tag -d 名字 #删除标签
git tag commit_id
```

还可以创建带有说明的标签

```
-a 标签名 -m 注释说明
git tag -a v1.0 -m '第一次打标签'
```

## (二)、操作标签

```
git tag -d tag-name #删除标签
git push origin v1.0 #推送标签到远程库中
git push origin --tags
```



The image shows a GitHub repository page for 'zhd-haodong / 0730study' on the left and a terminal window on the right. The GitHub page displays the 'Tags' section with three tags: v1.1 (created 14 minutes ago), v1.0\_tongbu (created 2 hours ago), and v1.2 (created yesterday). The terminal window shows the following commands and output:

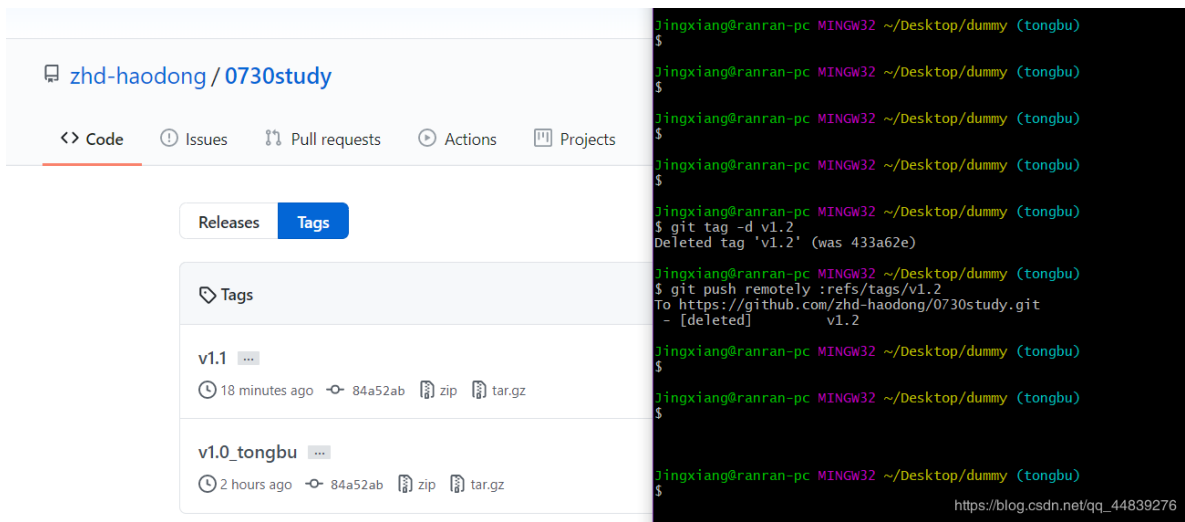
```
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$ git tag v1.2 433a
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$ git tag
v1.0_tongbu
v1.1
v1.2
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$ git push remotely v1.2
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/zhd-haodong/0730study.git
 * [new tag]         v1.2 -> v1.2
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$ git push remotely --tags
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/zhd-haodong/0730study.git
 * [new tag]         v1.0_tongbu -> v1.0_tongbu
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$
Jingxiang@ranran-pc MINGW32 ~/Desktop/dummy (tongbu)
$
https://blog.csdn.net/qq_44839276
```

如果标签已经推送到远程, 那么怎么删除呢??  
先把本地的删除

```
git tag -d v1.2
```

然后删除远程的, 用push命令

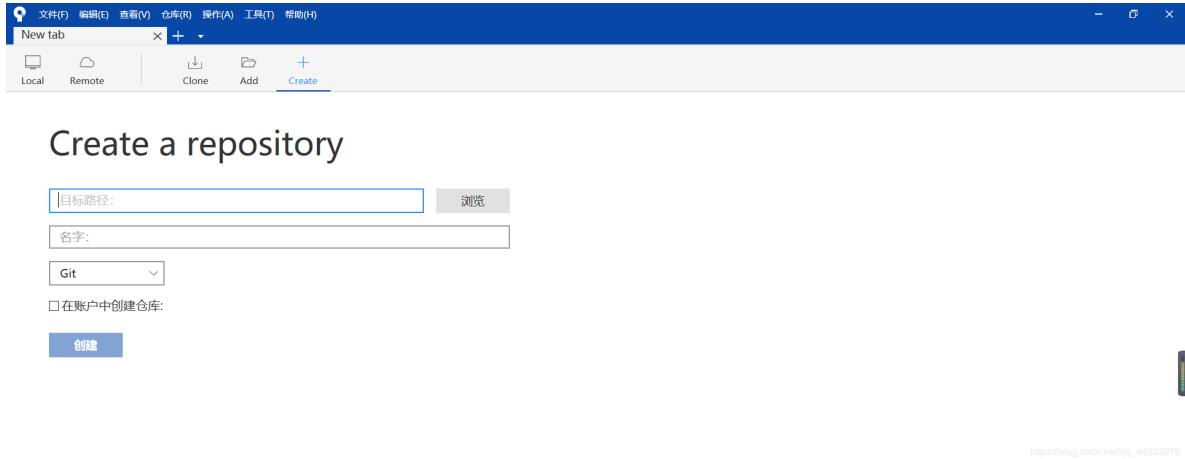
```
git push remotely :refs/tags/v1.2
```



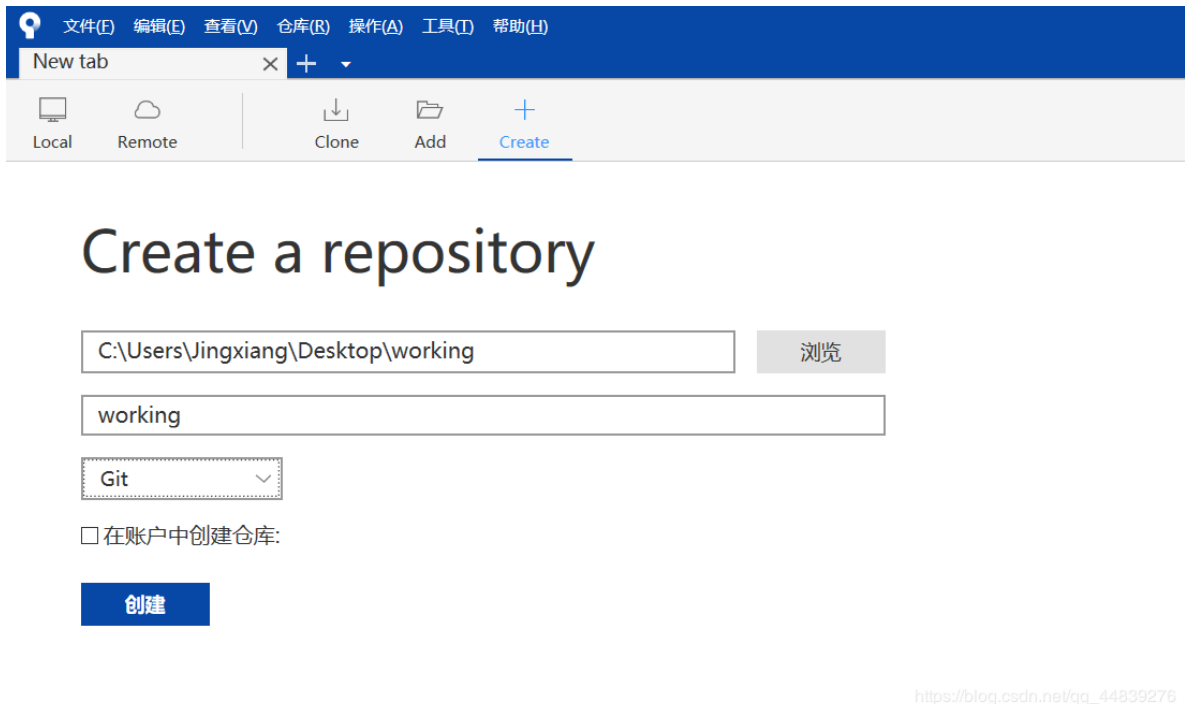
大功告成

## 九、Sourcetree助你一臂之力

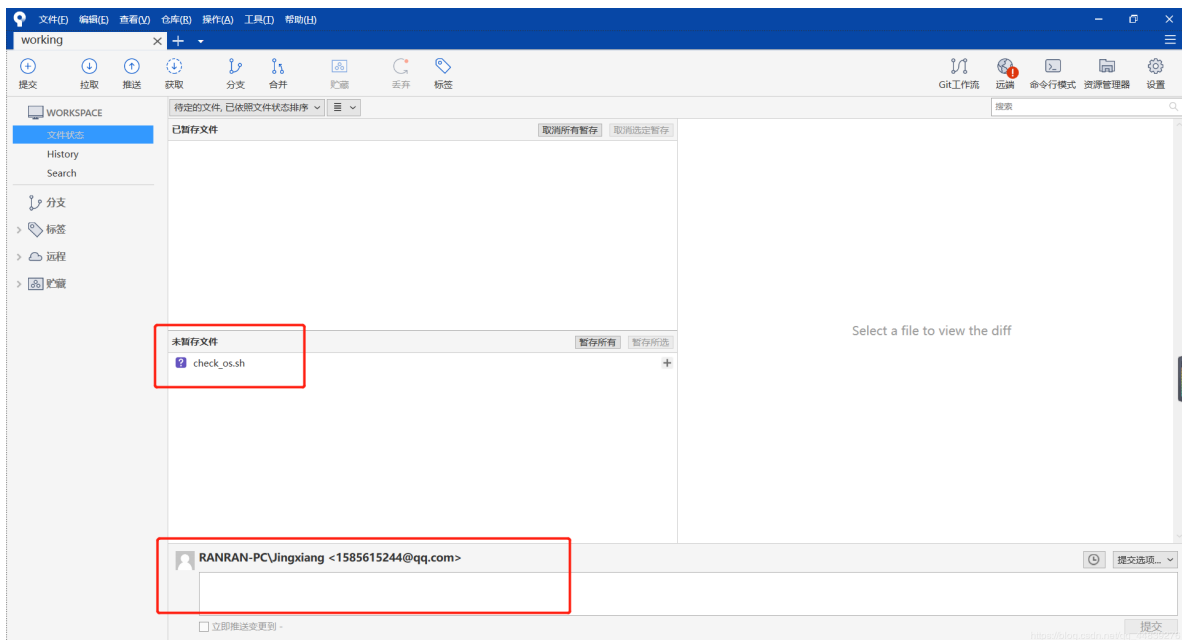
### 1、网上自行搜索下载sourcetree（很多很多）



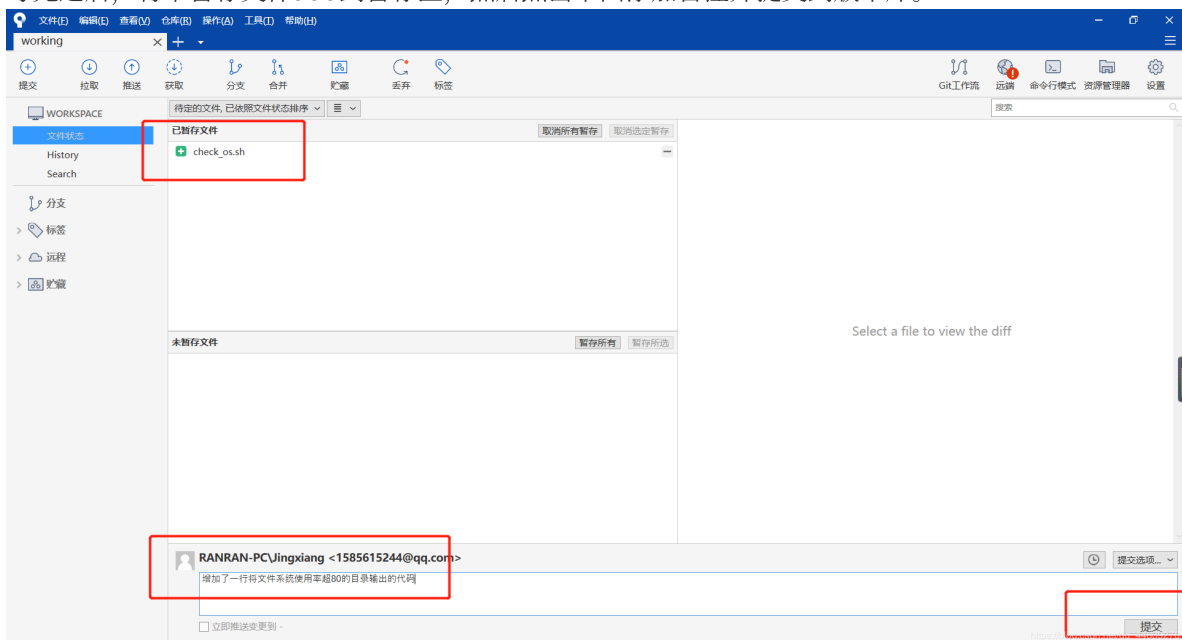
### 2、下面创建仓库



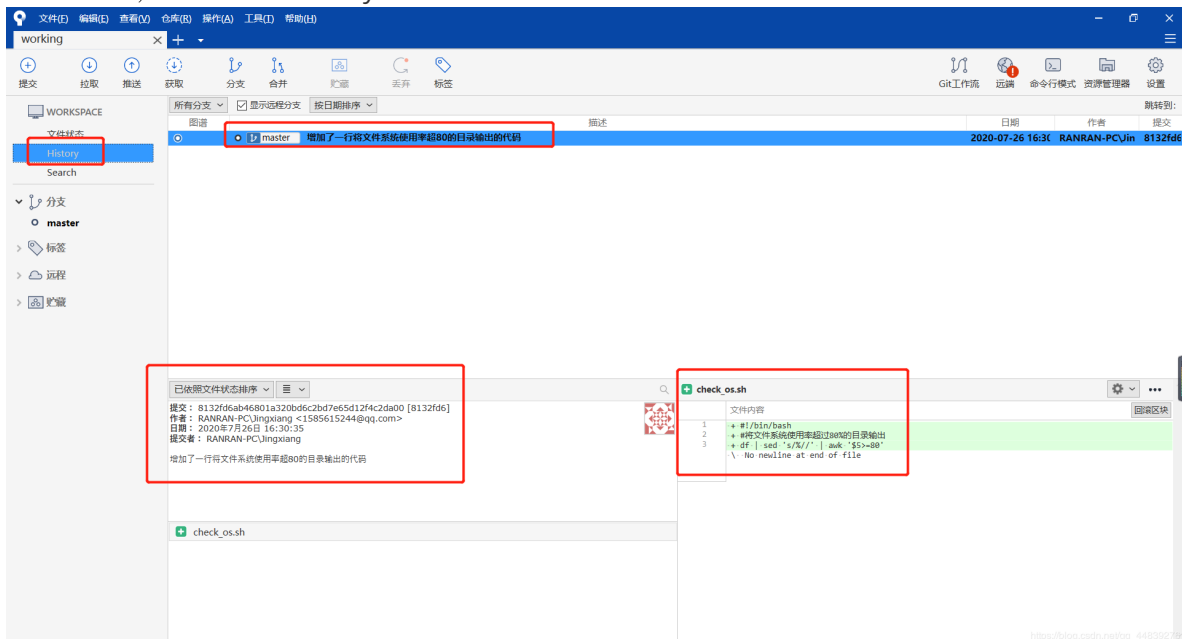
### 3、创建完之后，点击在文件资源管理器中打开，新建一个文本，开始写自己的代码吧。



写完之后，将未暂存文件add到暂存区，然后点击下面添加备注并提交到版本库。

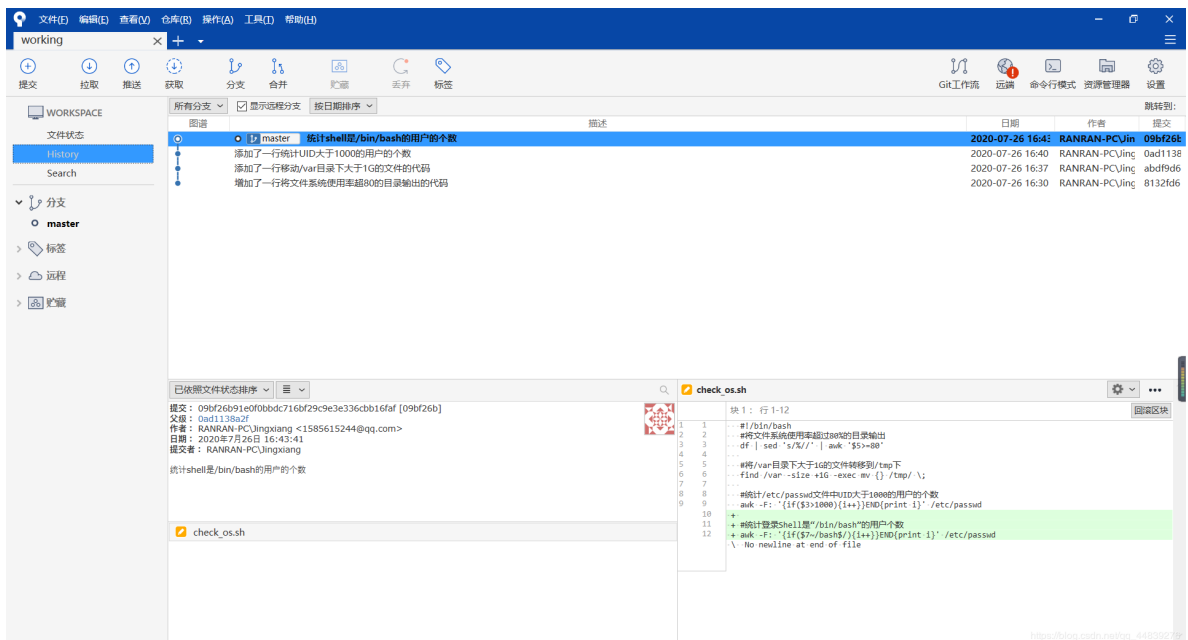


提交完之后，就可以点击history进行查看了。



下面我们先多些几次代码并提交，最后的效果如下：





我修改了四次代码，每次代码的注释都写了，最终代码的文件是这样的

```
#!/bin/bash
#将文件系统使用率超过80%的目录输出
df | sed 's/%//' | awk '$5>=80'

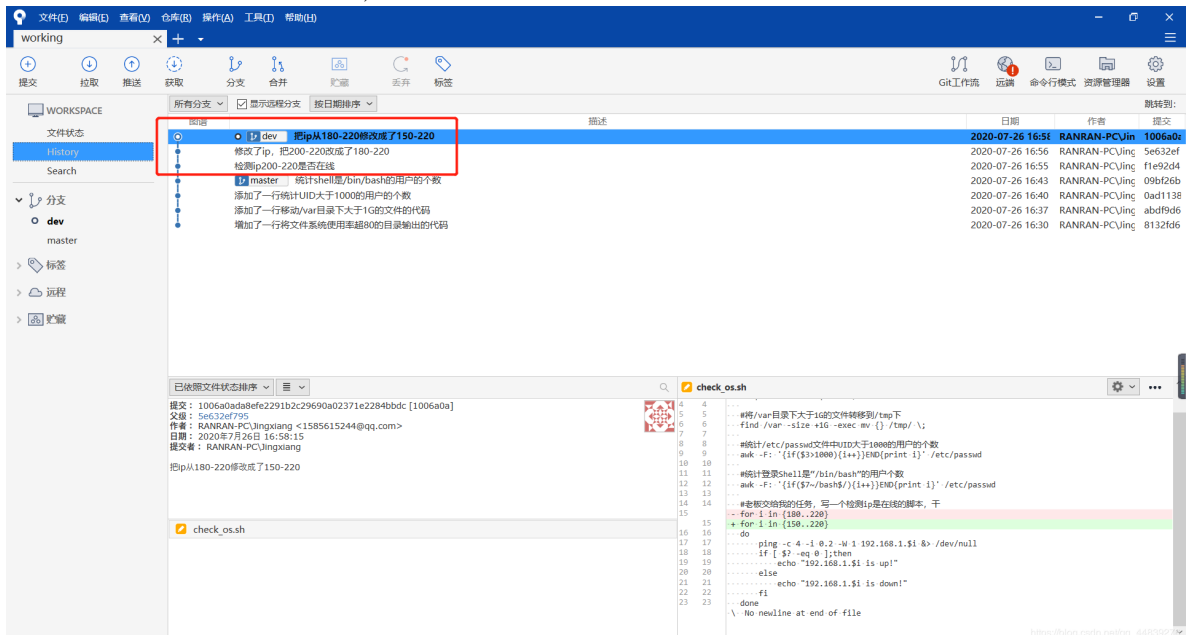
#将/var目录下大于1G的文件转移到/tmp下
find /var -size +1G -exec mv {} /tmp/ \;

#统计/etc/passwd文件中UID大于1000的用户的个数
awk -F: '{if($3>1000){i++}}END{print i}' /etc/passwd

#统计登录Shell是"/bin/bash"的用户个数
awk -F: '{if($7~/bash$/{i++}}END{print i}' /etc/passwd
```

现在有了新的需求了，我需要另一个同事给我写一个检测ip地址是否在线的脚本，那么这个同事就得新建一个分支来写脚本，以为我还有我的工作要做啊，他把脚本写完了告诉我一合并就ok了，这就是共同协作共同开发。

现在新建一个分支名字叫做dev，这位同事将在这个分支里完成我交给他的任务。



这位同事袖肥并写完了ip检测的脚本，内容如下：

```

Jingxiang@ranran-pc MINGW32 ~/Desktop/working (dev)
$ cat check_os.sh
#!/bin/bash
#将文件系统使用率超过80%的目录输出
df | sed 's/%//' | awk '$5>=80'

#将/var目录下大于1G的文件转移到/tmp下
find /var -size +1G -exec mv {} /tmp/ \;

#统计/etc/passwd文件中UID大于1000的用户的个数
awk -F: '{if($3>1000){i++}}END{print i}' /etc/passwd

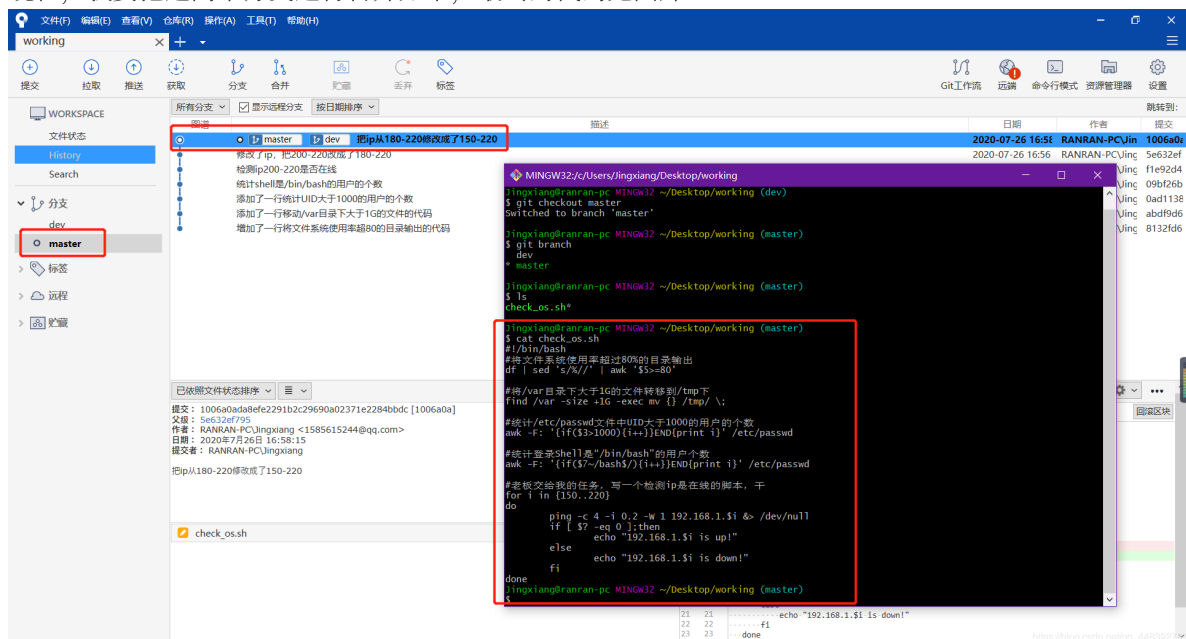
#统计登录Shell是"/bin/bash"的用户个数
awk -F: '{if($7~/bash$/){i++}}END{print i}' /etc/passwd

#老板交给我的任务，写一个检测ip是在线的脚本，干
for i in {150..220}
do
    ping -c 4 -i 0.2 -w 1 192.168.1.$i &> /dev/null
    if [ $? -eq 0 ];then
        echo "192.168.1.$i is up!"
    else
        echo "192.168.1.$i is down!"
    fi
done
Jingxiang@ranran-pc MINGW32 ~/Desktop/working (dev)
$ |

```

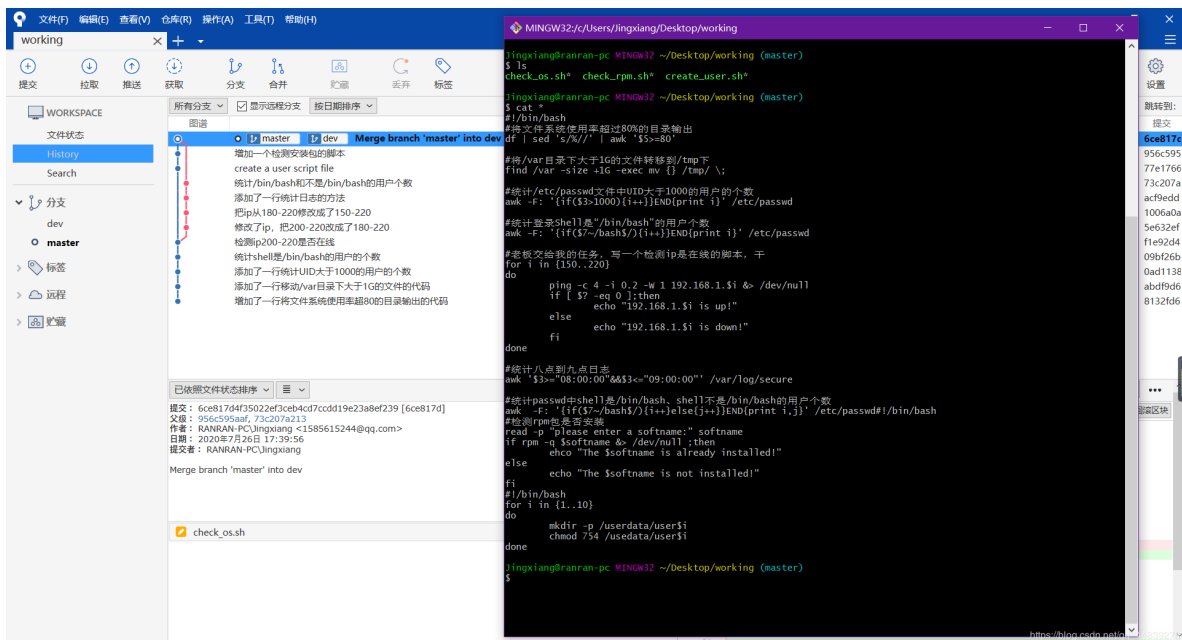
[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

现在，我要把这两个分支进行合并如下，最终的代码见图片

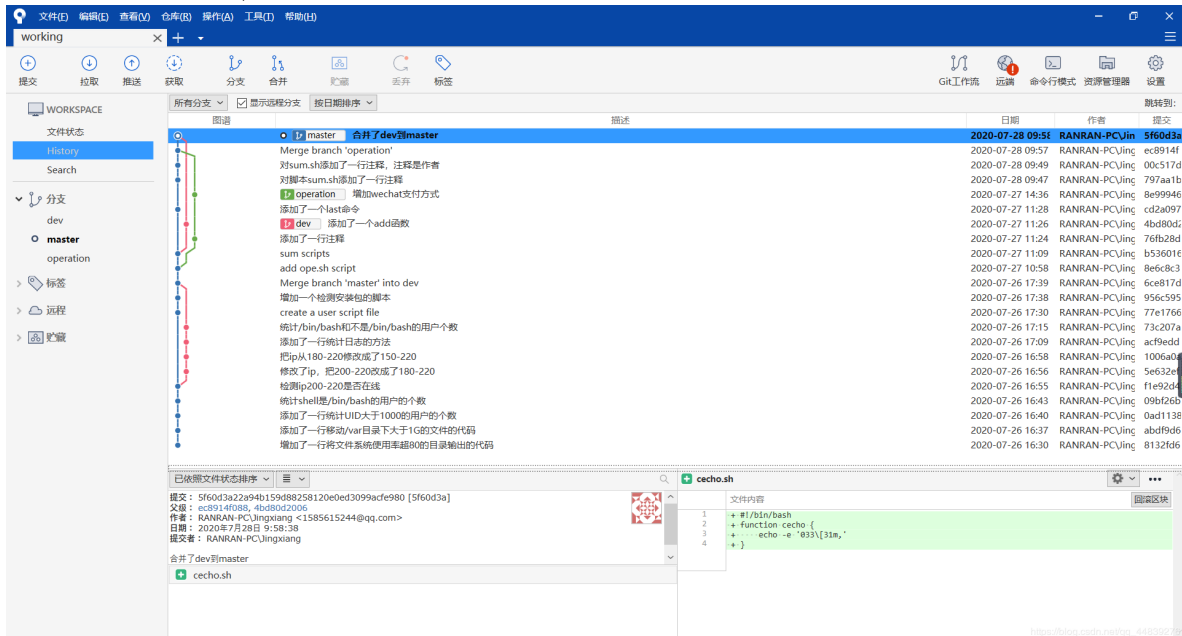


下面这张图，红色的是我在dev这个分支里面又写了两个文件，然后我在master分支里进行merge的操作。

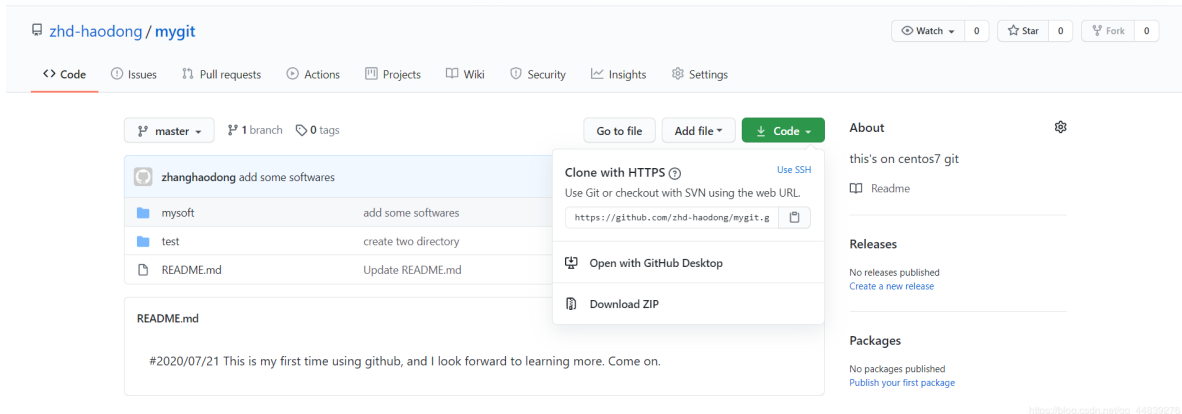
注意：在两个分支里，文件（文件名）相同的情况下，图谱只有一条蓝色的线，而当两个分支里的文件不同，一个分支有一个文件而另一个分支有两个以上的文件的时候才会有下面的情况，一条蓝线一条红线。



当有多条分支的时候，类似于下面：



在MINGW32下clone我github上的项目：



```
git clone https://github.com/zhd-haodong/mygit.git
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/working (master)
$ git clone https://github.com/zhd-haodong/mygit.git
Cloning into 'mygit'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 18 (delta 0), reused 15 (delta 0), pack-reused 0
Receiving objects: 100% (18/18), 480.05 KiB | 197.00 KiB/s, done.
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/working (master)
$ ls
cecho.sh*   check_rpm.sh*  mm.sh*   ope.sh*   wechat.sh*
check_os.sh* create_user.sh* mygit/   sum.sh*
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/working (master)
$ cd mygit/
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/working/mygit (master)
$ ls
mysoft/  README.md  test/
```

```
Jingxiang@ranran-pc MINGW32 ~/Desktop/working/mygit (master)
$ ls mysoft/
libxaw-1.0.13-4.e17.x86_64.rpm  xorg-x11-apps-7.7-7.e17.x86_64.rpm
```

[https://blog.csdn.net/qq\\_44839276](https://blog.csdn.net/qq_44839276)

## 十、远程仓库的使用之将本地关联到远程仓库上

下面介绍如何远程管理自己的仓库。

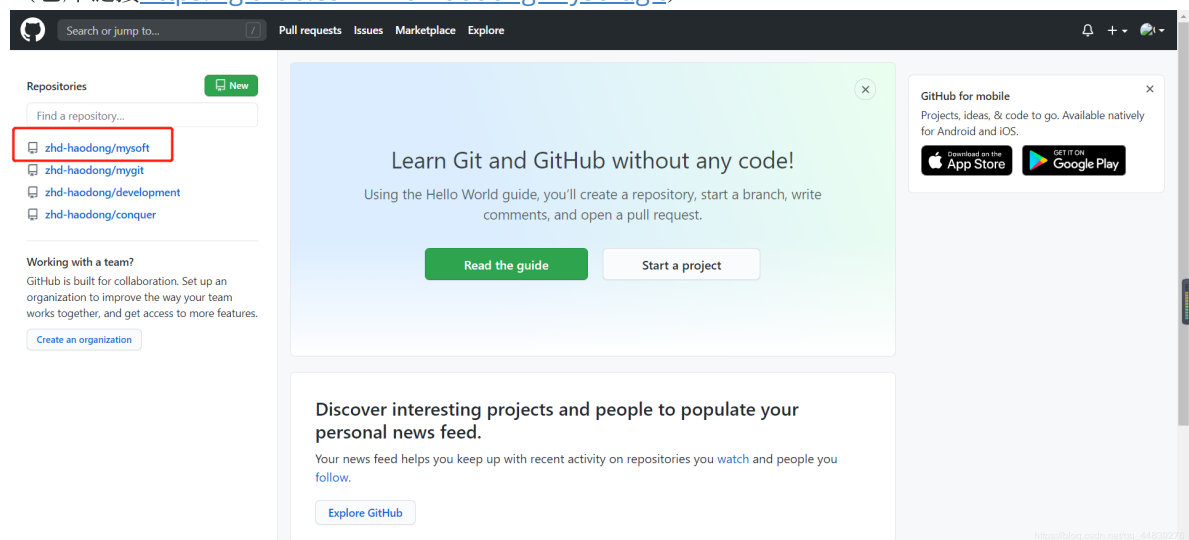
远程仓库指的就是在因特网或其他网络上的你的项目的版本库。我们可以建立好几个仓库，可读可写的都行，可以与他人一起管理我们的远程仓库，比如拉取推送等等。

管理远程仓库包括

- (1) 如何添加远程仓库
- (2) 移除现有的远程仓库，添加新仓库
- (3) 添加分支并同步到远程仓库里

### (一)、将本地文件同步到远程仓库（sourcetree和git命令行结合）

- (1) 在github上新建一个仓库如下（我新建的叫mysoft）  
(仓库链接<https://github.com/zhd-haodong/mysoft.git>)



- (2) 在要上传文件的根目录下，打开git命令窗口，键入一下命令

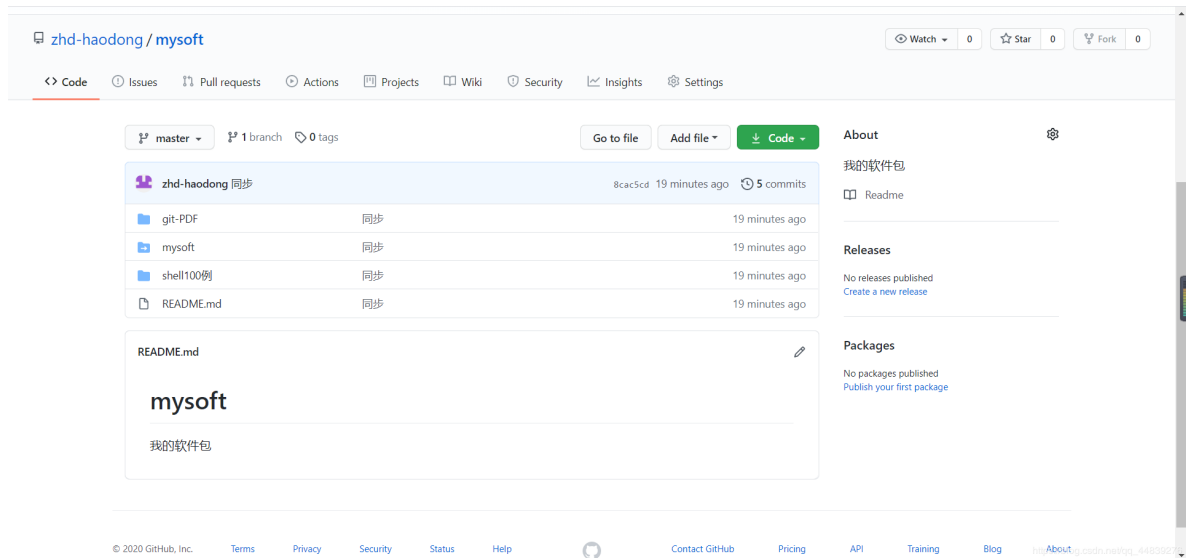
```

1) git init
2) git add .
3) git commit -m '第一次提交内容'
4) git关联远程仓库(origin是我们给远程仓库取得名字, 比较专业哈)
git remote add origin https://github.com/zhd-haodong/mysoft.git
5) 将本地文件与远程仓库进行合并(此步骤可以省略)
git pull --rebase origin master
6) 将本地文件推送到远程仓库中
git push -u origin master

```

把本地的内容推送到远程仓库上, 用git push, 实际上就是把当前的分支master推送到了远程上。由于我们新建的远程仓库是空的, 第一次推送master分支时, 使用了-u参数, 这样git不但会把本地的master分支推送到远程上, 还会把本地的master分支和远程的master分支进行关联, 在以后pull或push时就会简化命令了。

(3) 最后远程仓库上查看一下是否和本地一样



所以现在来说, 只要是你提交完毕之后, 就可以运行以下命令进行同步了

```
git push origin master
```

## (二)、移除现有的远程仓库, 添加新仓库

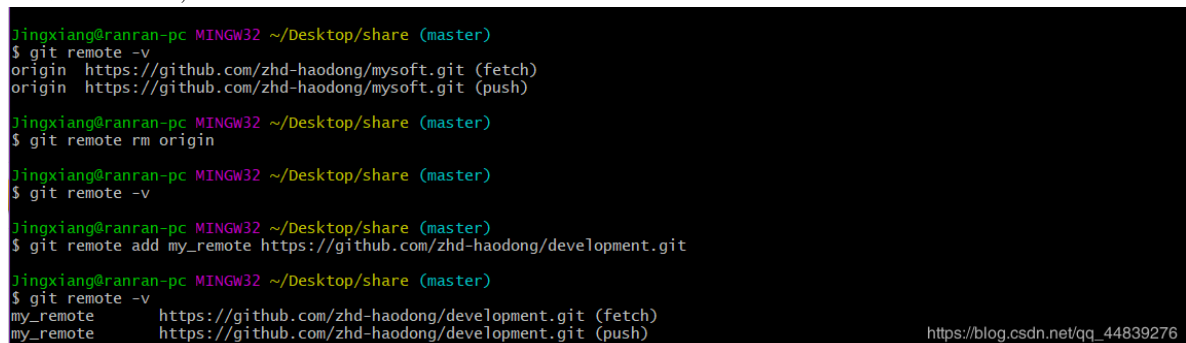
(1) 常用命令

```

git remote -v #查看现有远程仓库
git remote rm 名字 #删除现有仓库
git remote add 名字 https://xxxx/xx.git #添加新仓库

```

(2) 删除旧的, 添加新的

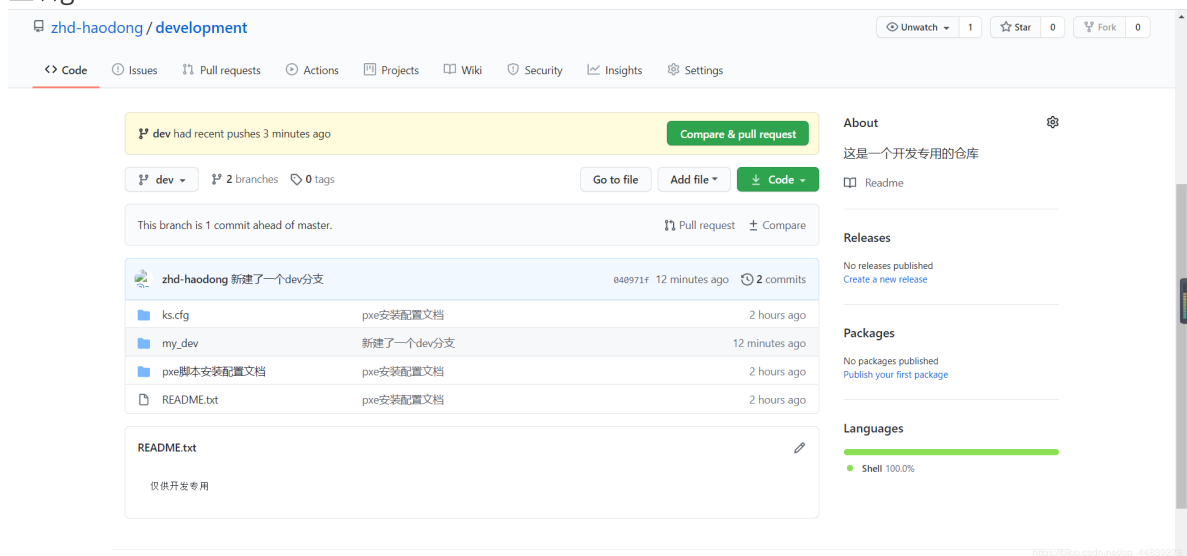


## (三)、添加分支，并同步到远程仓库上

```
git branch -b dev
mkdir my_dev
git push origin dev
```

```
dingxiang@ranran-pc MINGW32 ~/Desktop/development (dev)
$ ls
ks.cfg/  my_dev/  pxe脚本安装配置文档/  README.txt
dingxiang@ranran-pc MINGW32 ~/Desktop/development (dev)
$ git add .
warning: LF will be replaced by CRLF in my_dev/test.
the file will have its original line endings in your working directory
dingxiang@ranran-pc MINGW32 ~/Desktop/development (dev)
$ git commit -m '新建了一个dev分支'
[dev 040911f] 新建了一个dev分支
1 file changed, 1 insertion(+)
create mode 100644 my_dev/test
dingxiang@ranran-pc MINGW32 ~/Desktop/development (dev)
$ git remote -v
origin https://github.com/zhd-haodong/development.git (fetch)
origin https://github.com/zhd-haodong/development.git (push)
dingxiang@ranran-pc MINGW32 ~/Desktop/development (dev)
$ git push origin dev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 423 bytes | 423.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/zhd-haodong/development/pull/new/dev
remote:
To https://github.com/zhd-haodong/development.git
 * [new branch]      dev -> dev
```

### 查看github



## (四)、小结

### (1) 关联远程仓库使用命令

```
git remote add origin git@server-name:path/reponame.git
```

### (2) 关联后进行第一次推送master内容

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

### (3) 以后只要你提交了，就可以进行推送实现同步了

```
git push origin master
```

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！

本文借鉴自<https://www.liaoxuefeng.com/wiki/896043488029600>，谢谢！