

## 目录

- 1. 远程仓库-GitHub
  - 1.1 本地电脑如何关联GitHub?
  - 1.2. 创建并操控远程库GitHub
    - (1) 把一个已有的本地仓库与之关联, 然后, 把本地仓库的内容推送到GitHub仓库。: 本地->远程
    - (2) 从GitHub克隆库到本地: 远程->本地
- 2. Git分支管理(重要)
  - 2.1. 创建和合并分支
    - 1) 增加分支的原理
    - 2) 增加分支后的提交变化
    - 3) 分支合并
    - 4) 分支删除
    - 代码测试
  - 2.2. 解决合并冲突
  - 2.3. 分支管理策略
  - 2.4. Bug分支
    - 1) git stash
    - 2) 创建issue分支解决bug
      - 1) 恢复办法1: git stash apply
      - 2) 恢复方法2: git stash pop
  - 2.5. Feature分支

# 1. 远程仓库-GitHub

Git是分布式版本控制系统, 同一个Git仓库, 可以分布到不同的机器上。怎么分布呢? 最早, 肯定只有一台机器有一个原始版本库, 此后, 别的机器可以“克隆”这个原始版本库, 而且每台机器的版本库其实都是一样的, 并没有主次之分。

实际情况往往是这样, 找一台电脑充当服务器的角色, 每天24小时开机, 其他每个人都从这个“服务器”仓库克隆一份到自己的电脑上, 并且各自把各自的提交推送到服务器仓库里, 也从服务器仓库中拉取别人的提交。

## 1.1 本地电脑如何关联GitHub?

由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的, 所以, 需要一点设置:

### • 创建SSH Key

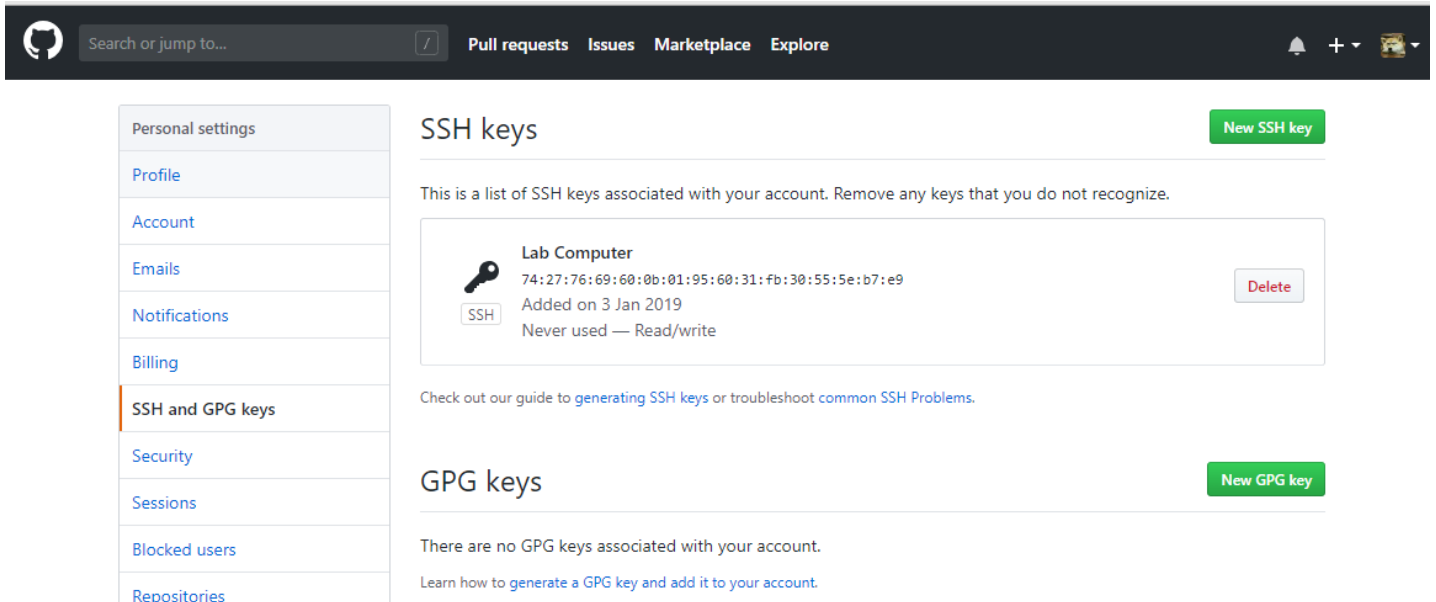
```
$ ssh-keygen -t rsa -C "haochen273@gmail.com"
```

一路回车就可以了

如果一切顺利的话, 可以在用户主目录 (C:\Users\haoch.ssh) 里找到 `.ssh` 目录, 里面有 `id_rsa` 和 `id_rsa.pub` 两个文件, 这两个就是 `SSH Key` 的密钥对, `id_rsa` 是私钥, 不能泄露出去, `id_rsa.pub` 是公钥, 可以放心地告诉任何人。

### • 设置GitHub的SSH Key

在GitHub主页找到 `Account Seetings`, `SSH Key` 页面: 将 `id_rsa.pub` 的内容粘贴到里面即可Add



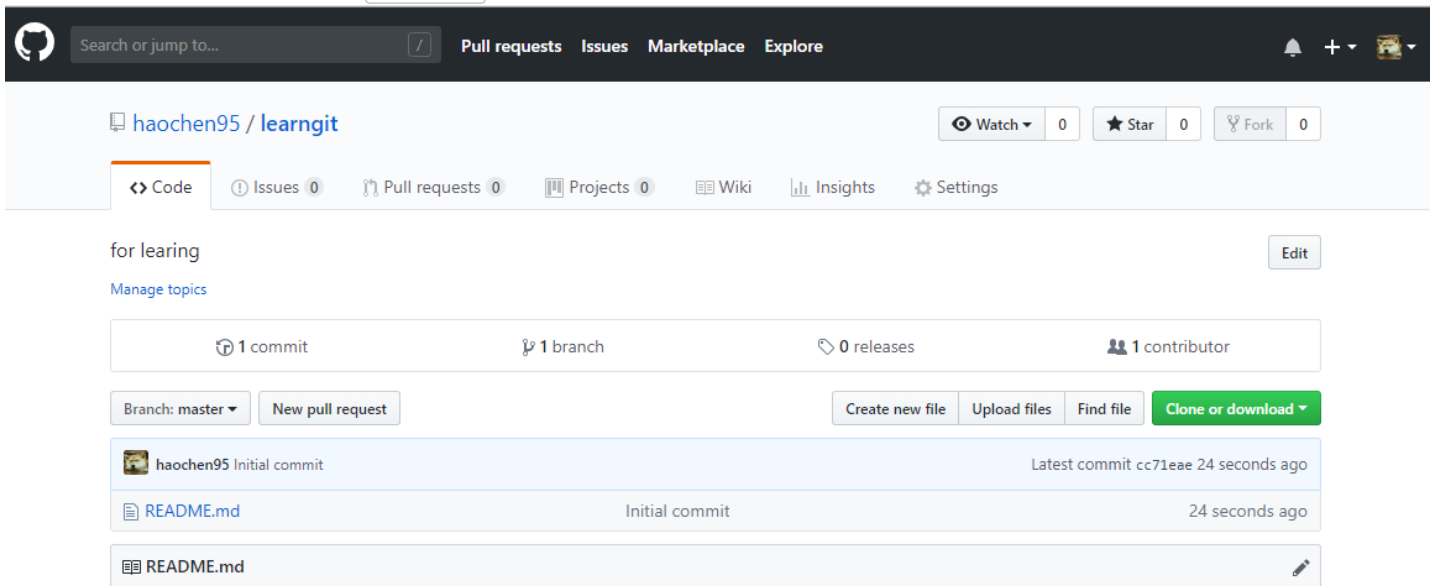
### GitHub为什么要用SSH Key呢?

因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

## 1.2. 创建并操控远程库GitHub

在GitHub中创建一个远程库名为 `learngit`：



**(1) 把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。：本地->远程**

- **关联**

在本地库运行命令：

```
$ git remote add origin git@github.com:haochen95/learngit.git
```

添加后，远程库的名字就是 `origin`，这是Git默认的叫法，也可以改成别的，但是 `origin` 这个名字一看就知道是远程库。

- **推送**

使用命令 `git push -u origin master -f`

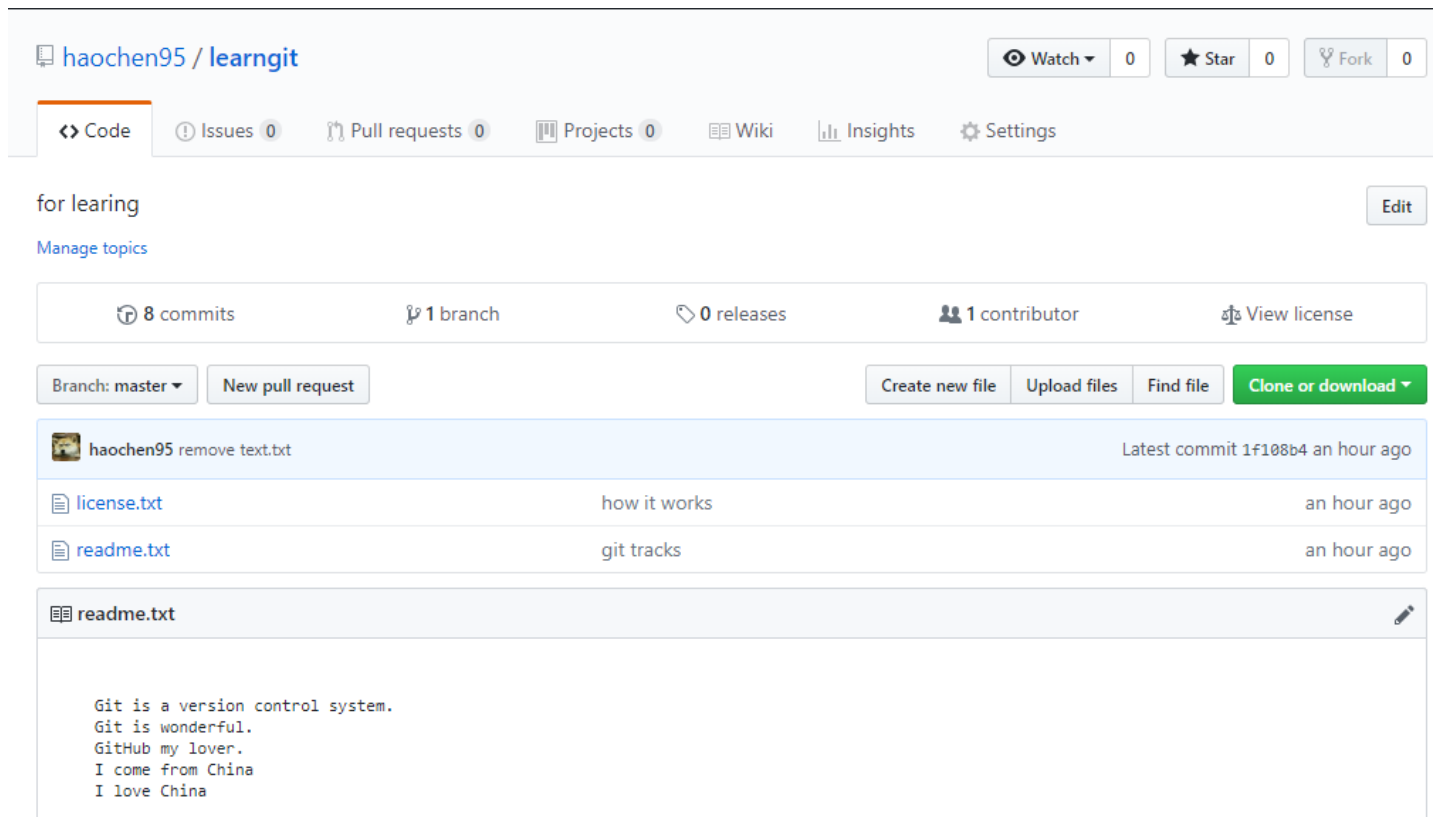
```
$ git push -u origin master -f
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (17/17), done.
Writing objects: 100% (23/23), 1.82 KiB | 465.00 KiB/s, done.
Total 23 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To github.com:haochen95/learngit.git
+ cc71eae...1f108b4 master -> master (forced update)
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

把本地库的内容推送到远程，用 `git push` 命令，实际上是把当前分支 `master` 推送到远程。

由于远程库是空的，我们第一次推送 `master` 分支时，加上了 `-u` 参数，Git 不但会把本地的 `master` 分支内容推送的远程新的 `master` 分支，还会把本地的 `master` 分支和远程的 `master` 分支关联起来，在以后的推送或者拉取时就可以简化命令。

成功后的页面



haochen95 / learngit

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

for learning Edit

Manage topics

8 commits 1 branch 0 releases 1 contributor View license

Branch: master New pull request Create new file Upload files Find file Clone or download

haochen95 remove text.txt		Latest commit 1f108b4 an hour ago
license.txt	how it works	an hour ago
readme.txt	git tracks	an hour ago

readme.txt

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
I love China
```

- 以后的每次提交只写一个命令

```
$ git push origin master
```

把本地 `master` 分支的最新修改推送至 `GitHub`，现在，你就拥有了真正的分布式版本库！

## (2) 从GitHub克隆库到本地: 远程->本地

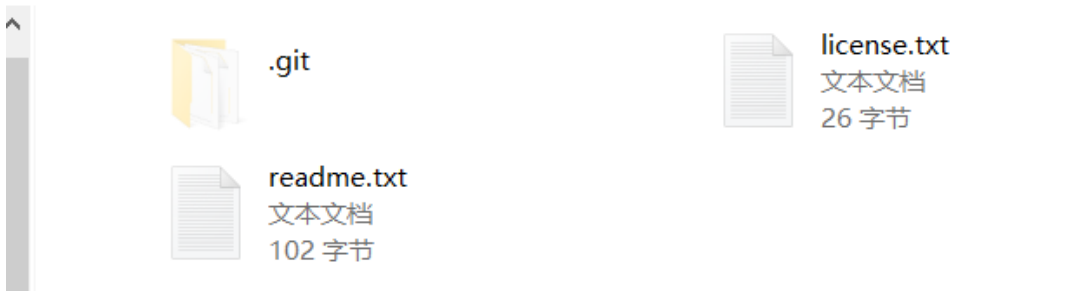
现在我们另外创建一个空文件夹: `D:\git_clone`

克隆的代码是: `$ git clone git@github.com:haochen/learngit.git`

```
$ git clone git@github.com:haochen95/learngit.git
Cloning into 'learngit'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (12/12), done.
Receiving objects: 100% (23/23), done.
Resolving deltas: 100% (5/5), done.
remote: Total 23 (delta 5), reused 23 (delta 5), pack-reused 0
```

你看，本地的库里面就有了跟GitHub一样的内容

此电脑 > 新加卷 (D:) > git\_clone > learngit



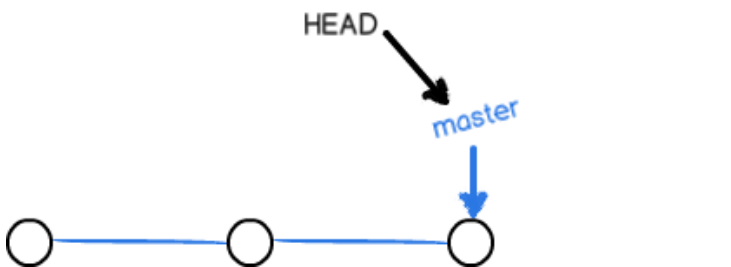
## 2. Git分支管理(重要)

创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

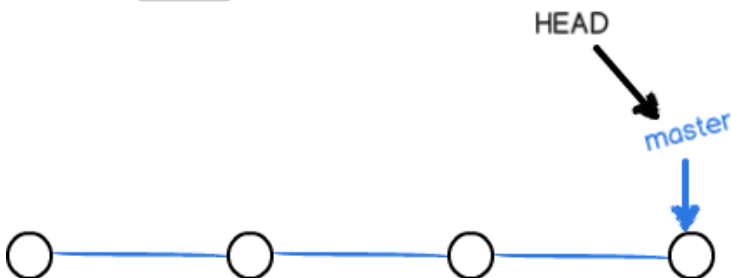
### 2.1. 创建和合并分支

在 版本回退 里，你已经知道，每次提交，Git 都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在 Git 里，这个分支叫主分支，即 master 分支。HEAD 严格来说不是指向提交，而是指向 master，master 才是指向提交的，所以，HEAD 指向的就是当前分支

一开始的时候，master 分支是一条线，Git 用 master 指向最新的提交，再用 HEAD 指向 master，就能确定当前分支，以及当前分支的提交点：

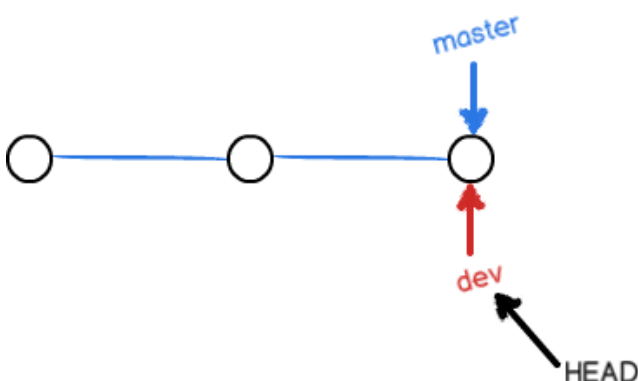


每次提交，master 分支都会向前移动一步，这样，随着你不断提交，master 分支的线也越来越长：



#### 1)增加分支的原理

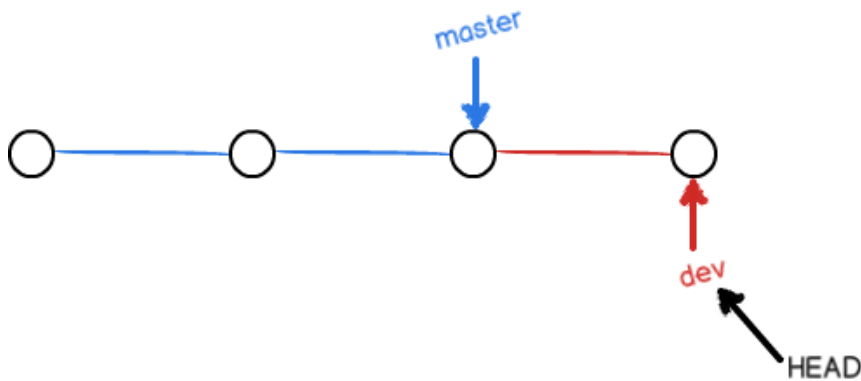
我们创建新的分支，例如 dev 时，Git 新建了一个指针叫 dev，指向 master 相同的提交，再把 HEAD 指向 dev，就表示当前分支在 dev 上



**增加分支的原理：** 增加一个指针，更改 `HEAD` 的指向

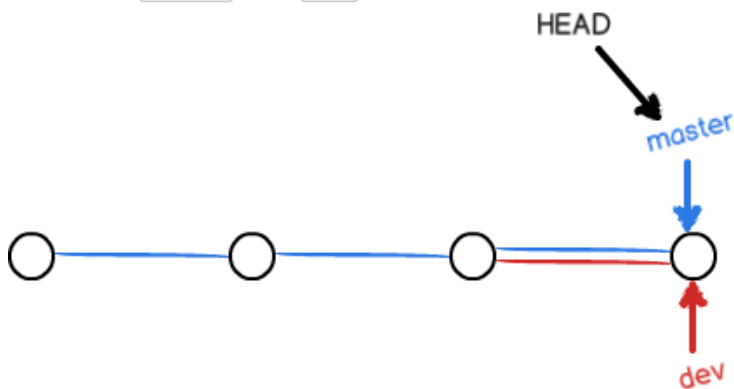
## 2) 增加分支后的提交变化

现在开始，对工作区的修改和提交就是针对 `dev` 分支了，比如新提交一次后，`dev` 指针往前移动一步，而 `master` 指针不变：



## 3) 分支合并

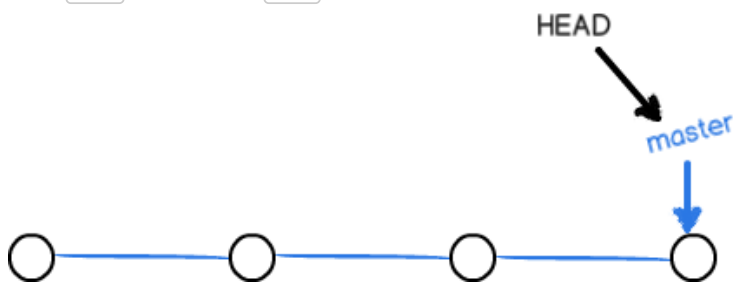
就是直接把 `master` 指向 `dev` 的当前提交，就完成了合并



**合并分支的原理：** 改改指针 (`dev->master`)

## 4) 分支删除

删除 `dev` 分支就是把 `dev` 指针给删掉，删掉后，我们就剩下了一条 `master` 分支



## 代码测试

### 1. 创建分支

```
$ git checkout -b dev
Switched to a new branch 'dev'
```

`git checkout` 命令加上 `-b` 参数表示创建并切换

### 2. 查看分支

```
$ git branch
* dev
master
```

### 3. 在分支上提交

增加一个文件命名为 `love.txt` 并且提交

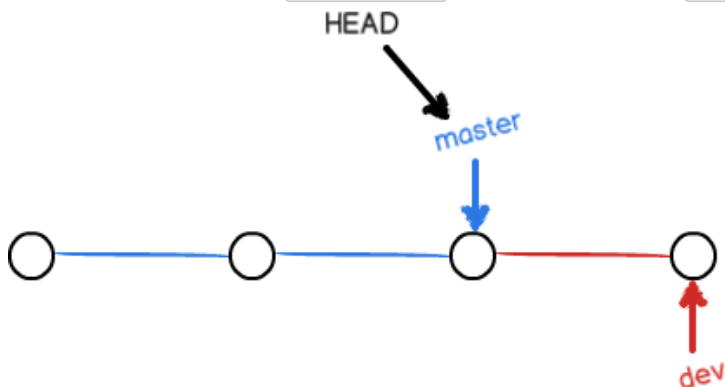
```
$ git add love.txt

$ git commit -m "love"
[dev b3045f0] love
1 file changed, 1 insertion(+)
create mode 100644 love.txt
```

#### 4. 从dev分支切换回master分支

```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

再次查看文件夹发现没有 love.txt 文件，因为那个提交是在 dev 分支上，而 master 分支此刻的提交点并没有变



#### 5. master和dev分支合并

```
$ git merge dev
Updating 1f108b4..b3045f0
Fast-forward
 love.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 love.txt
```

#### 6. 删除dev分支

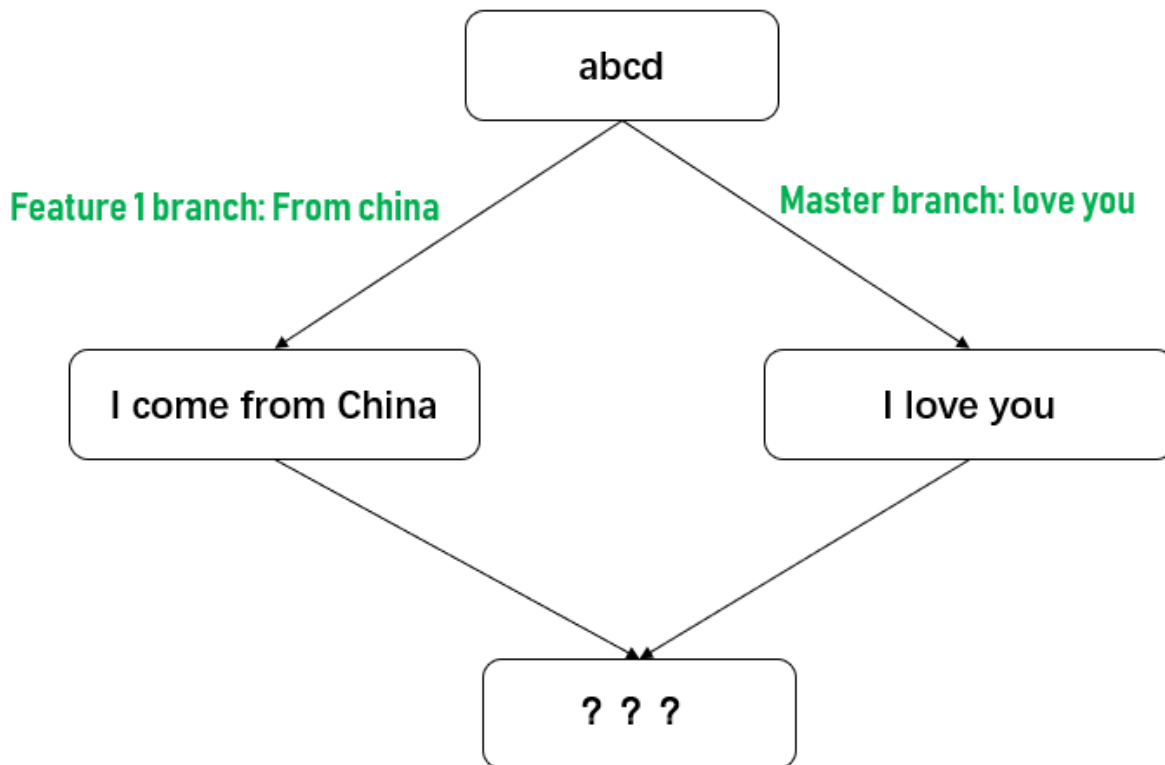
```
$ git branch -d dev
Deleted branch dev (was b3045f0).
```

#### 7. 再次查看branch信息

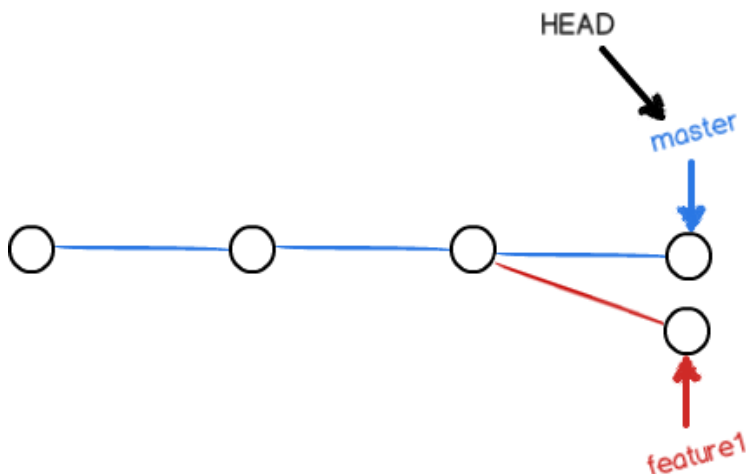
```
$ git branch
* master
```

## 2.2. 解决合并冲突

问题：在分支上提交一个修改，转到master后又提交一个修改，两个修改都在同一个位置，请问如何合并？



现在，`master` 分支和 `feature1` 分支各自都分别有新的提交，变成了这样：



这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看

```
$ git merge feature1
Auto-merging love.txt
CONFLICT (content): Merge conflict in love.txt
Automatic merge failed; fix conflicts and then commit the result.
```

需要手动解决冲突

通过 `git status` 查看状态

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

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:   love.txt

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

我们可以直接查看 `love.txt` 的内容:

```
<<<<<< HEAD
I love you
=====
I come from china
>>>>>> feature1
```

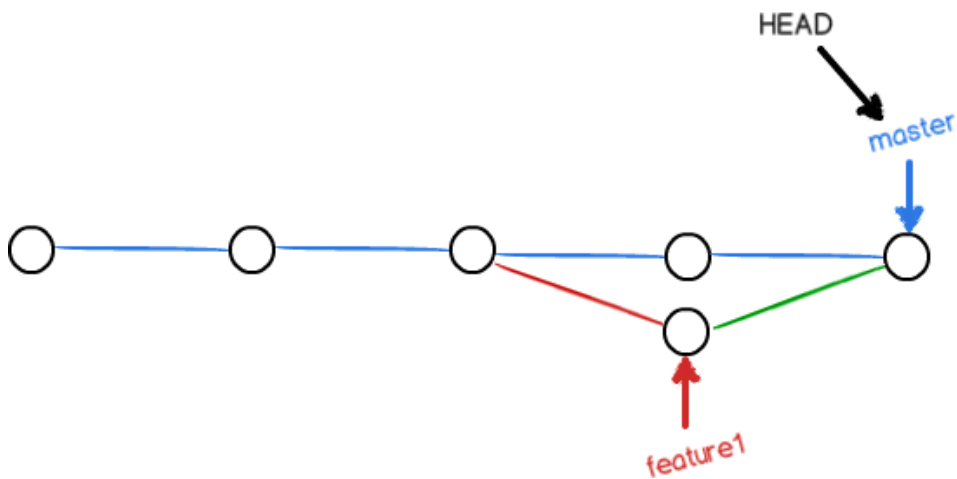
我们重新修改为

```
I come from china
```

然后再提交

```
$ git commit -m "loving"
[master e6c7c5f] loving
```

现在, master分支和feature1分支变成了下图所示:



我们可以通过命令 `$ git log --graph --pretty=oneline --abbrev-commit` 查看分支情况

```
$ git log --graph --pretty=oneline --abbrev-commit
* e6c7c5f (HEAD -> master) loving
| \
| * 445380a (feature1) from china
* | 62b6881 love you
| /
* b3045f0 love
* 1f108b4 (origin/master) remove text.txt
* f2863ca new text
* aa4f64e git tracks
* 45700b2 how it works
* 5103166 delete software
* 9fbb435 github infor
* 4fb84da add new line
* f7f8050 write to readme
```

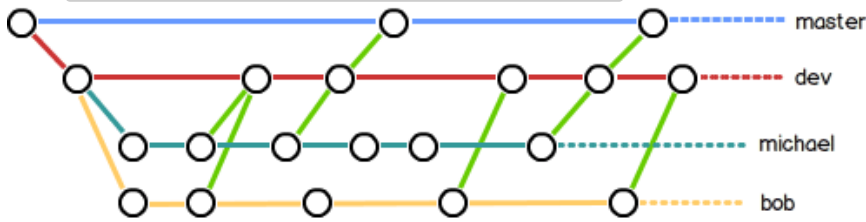
## 2.3. 分支管理策略

1. `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;
2. 干活都在 `dev` 分支上, 也就是说, `dev` 分支是不稳定的, 到某个时候, 比如1.0版本发布时, 再把 `dev` 分支合并到 `master` 上, 在 `master` 分支发布1.0版本



3. 你和你的小伙伴们每个人都在 `dev` 分支上干活, 每个人都有自己的分支, 时不时地往 `dev` 分支上合并就可以了。

4. 使用 `git merge --no-ff -m "merge with no-ff" dev` 合并会留下历史信息, 建议这样用



## 2.4. Bug分支

假设, 你正正在完成 `love.txt` 的编辑工作,但是这个文件需要2天才能完成, 但是现在老板给你说 `readme.txt` 有一个错误(I love USA)你需要更改这个bug, 那怎么办? 你想创建一个分支 `issue-101` 来修复它, 但是, 等等, 当前正在 `feature1` 上进行的工作还没有提交:

```
$ git status
On branch feature1
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   love.txt

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
```

### 1) `git stash`

这个功能是把当前工作现场“储藏”起来, 等以后恢复现场后继续工作:

```
$ git stash
Saved working directory and index state WIP on feature1: f1f394f modified
```

现在, 用 `git status` 查看工作区, 就是干净的 (除非有没有被Git管理的文件), 因此可以放心地创建分支来修复bug。

### 2) 创建issue分支解决bug

在master上创建分支

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 4 commits.
  (use "git push" to publish your local commits)

$ git checkout -b issue
Switched to a new branch 'issue'
```

然后修改 `readme.txt` 的删除I love China, 在提交

```
$ git commit -m "fix issue101"
[issue bdc0b2e] fix issue101
 1 file changed, 1 deletion(-)
```

然后转到master合并分支

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 4 commits.
  (use "git push" to publish your local commits)
```

```
$ git merge --no-ff -m "merged bug fix 101" issue
Merge made by the 'recursive' strategy.
 readme.txt | 1 -
 1 file changed, 1 deletion(-)
```

哈哈，Bug解决啦，是时候回到 `feature1` 分支继续工作啦

```
$ git checkout feature1
Switched to branch 'feature1'

$ git status
On branch feature1
nothing to commit, working tree clean
```

工作区是干净的，刚才的工作现场存到哪去了？用 `git stash list` 命令看看

```
$ git stash list
stash@{0}: WIP on feature1: f1f394f modified
```

工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法

**1)恢复办法1:** `git stash apply`

用 `git stash apply` 恢复，但是恢复后，`stash` 内容并不删除，你需要用 `git stash drop` 来删除；

**2)恢复方法2:** `git stash pop`

`git stash pop`，恢复的同时把stash内容也删了

## 2.5. Feature分支

添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添加一个新功能，最好新建一个 `feature` 分支，在上面开发，完成后，合并，最后，删除该 `feature` 分支。