## 具体流程：

git init **-> git remote add origin… -> git pull origin master + dev**

**-> 如果pull遇到了错误，提示（fetal :refusing to merge unrelated histories）则执行：git pull --rebase origin master -> git add . -> git commit -> git pull origin master / git pull origin dev -> git add . ->**

**Git commit … -> git push origin HEAD:dev**

## Clone 具体流程

### 在master上提交

新建文件夹 -》 git init -> git clone … -> (可能需要cd进入文件) -> Coding ->

Git add . -> git commit … -> git push origin master

### 在其他分支上提交

1. 新建分支可以直接提交
2. 提交到已经存在的分支：git pull origin dev -> git push origin HEAD:dev

## 操作具体步骤：

1.cd 路径 进入当前目录

2.配置git基本操作

Git config --global user.name “”

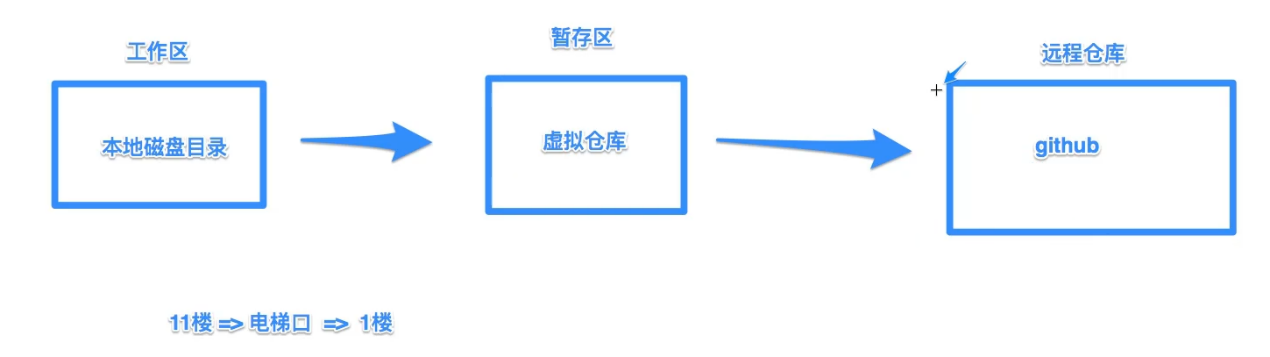
Git config --global user.email “”

【注】没有消息就是好消息

3. 1）建立远程仓库（github）

2）git init 在本地进行初始化（在本地建立建立一个暂存区）

.git文件存储当前项目的所有版本信息



4.工作区 =》暂存区（stage or index）

Git add 文件名

Git add \* 提交所有文件

Git commit -m “这一次提交的描述”

5.查看当前工作区的状态

Git status

6.撤销工作区的改变，

Git restore 文件名

与git checkout -- <>作用一样呢

7. 查看工作区和暂存区版本的区别

git diff

8.clear 清屏操作

9.git log

查看已经提交的历史版本

10.恢复文件到指定的某一版本

1）git reset --hard HEAD^ 退回到上个版本，好多版本就加好多^ eg:git reset --hard HEAD^^退回两个版本（但是这个有点问题，退回的是该版本的上个版本）

**实际上，git就是将版本链表的HEAD指针指向了前一个版本，但是版本2并没有被删除。**

**那再如何从版本1回到版本2呢：**

方法1：在知道版本号的情况下git reset --hard 版本号

方法2：当重新打开终端不知道版本号的情况下，git reflog 查看操作的记录。

 head节点代表最新的commit  显示head 信息：git show HEAD

11.从暂存区提取到远程区，需要生成一个ssh密匙

Ssh-keygen -t rsa -C [1285507636@qq.com(github邮箱地址)](mailto:1285507636@qq.com(github邮箱地址))

12.windows电脑查找文件：我的电脑=》用户=》用户名=》.ssh（隐藏文件）=》xxx.pub（存有密匙）

13.配置密匙，settings=>ssh

14. 暂存区=》远程仓库

1)仓库地址：git remote add origin

2)git push -u origin master， 提交到远程服务器

git push origin HEAD:coding提交到指定分支

【注】第一次需要输入用户名和密码

【注】可以直接Git pull origin 分支名

3）当出现错误，不能push时，使用git pull --rebase origin master

origin就是一个名字，它是在你clone一个托管在Github上代码库时，git为你默认创建的指向这个远程代码库的**标签**， origin指向的是repository，master只是这个repository中默认创建的第一个branch。当你git push的时候因为origin和master都是默认创建的，所以可以这样省略，但是这个是bad practice，因为当你换一个branch再git push的时候，有时候就纠结了

## git中的origin

<https://blog.csdn.net/niexia_/article/details/79422859?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task>

# git远程分支代码拉取

<https://blog.51cto.com/13893093/2175764?source=dra>

具体过程

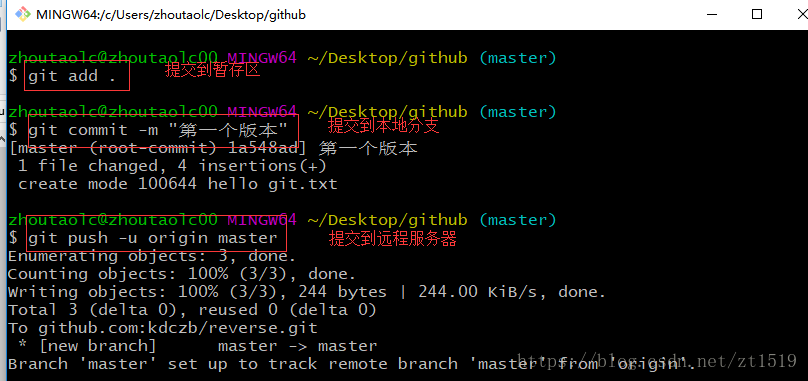
* 新建一个空文件
* 初始化 git init
* 自己要与origin master建立连接（下划线远程仓库链接）  
  git remote add origin <http://192.168.9.10:8888/root/game-of-life.git>
* 把远程分支拉到本地（game-of-live-first\_branch为远程仓库的分支名）  
  git fetch origin game-of-live-first\_branch
* 在本地创建分支game-of-live-first\_branch并切换到该分支  
  git checkout -b game-of-live-first\_branch origin/game-of-live-first\_branch
* 把game-of-live-first\_branch远程分支上的内容都拉取到本地  
  git pull origin game-of-live-first\_branch

15. git clone

从远程仓库克隆到本地

16. 从远程仓库直接拉项目到本地（同步本地代码），即更新，

Git pull



17. -m 表示modify,修改文件,-a表示add添加，执行删除后commit要加-a或者-am都可以

18.lunix命令

1. ls 表示当前文件夹下文件
2. Pwd 表示当前目录
3. Mkdir创建文件夹
4. 新建文件：touch name
5. Vi name（vi为打开编辑，不存在的会新建一个文件） 进入到文件，转换到插入状态，就可以编辑文本。若要退出，按esc，然后输入冒号：，然后w（write）q（quit），回车（或者直接x回车）；或者使用两次大写ZZ即可保存退出，使用cat name 来查看内容
6. 再说下正常退出，正常退出有个前提条件是：打开的文本文件在内容上没有被改动过。按了ESC后再输入冒号，在输入命令时，直接输入"q"
7. 不保存退出，很多时候打开了文件，或者修改了一些地方，才发现错了，非常需要不保存退出。先按ESC，再输入冒号，在输入命令时，直接输入"q!"
8. 强制退出。这个实在是不应该做的操作，因为很操蛋！先按ESC，再按冒号，在输入命令时，直接输入"!"，如图。但退出后，会有提示！

1.我们编辑文件是在工作区进行；  
2.git add 是将我们在工作区编辑好的文件提交给git，让git跟踪管理我们的代码；  
3.git commit是将我们放在暂存区的代码文件，创建一个版本记录，方便管理。

**git只会提交暂存区修改过的文件进行创建新版本。**  
所以：如果我们需要创建新版本，必须先把修改的文件添加到暂存区中。

## 如何撤销一个文件的修改：

**1.撤回还没有加入到暂存区的文件修改，也就是还在暂存区的文件：**

使用 “git checkout -- <文件>…” 丢弃工作区的改动：

**2.如何撤回已经添加到暂存区的文件:**

使用git reset HEAD <文件> 来撤销添加到暂存区操作：



git reset 把HEAD指向刚刚下载的最新的版本

* **git日志过多最后显示end无法输入命令**

点击q即可

* warning: LF will be replaced by CRLF in nalfne.

The file will have its original line endings in your working directory

执行：$ git config --global core.autocrlf false

1.初始化git仓库（提前创建好的目录）

git init

2.git配置

git config --global user.email "这里写你的邮箱"

git config --global user.name "这里写你的名字"

2.将一个文件添加到暂存区

git add filename

3.给暂存区的文件创建一个新版本(commit只给暂存区文件创建新版本)

git commit -m "版本说明信息"

4.查看文件状态

git status

5.查看提交日志，版本创建日志

git log

6.版本回退

git reset --hard 版本名 ， 如

git reset --hard HEAD^ 回退到上一个版本,实际上是让HEAD指针指向上一个版本

7.查看操作记录

git relog

8.丢弃工作区对某个文件的修改

git checkout --文件名

9.把提交到暂存区的文件撤回

git reset --hard 版本名 文件名 , 如

git reset --hard HEAD code.cpp ，将已经提交到暂存区的code.cpp文件撤出暂存区

10.将已经创建新版本的文件撤出

1. 版本回退
2. 撤出暂存区
3. 如果需要，丢失工作区对应文件的修改

* 比较工作区与HEAD指向的版本文件的不同

git diff HEAD --filename



+:代表工作区中的文件  
-:代表版本库中的文件，这里是HEAD指向的那个版本  
从上图可以直观的感受到，没有颜色的代表两个文件中共有的；  
+代表工作区中特有的语句

* 这里演示HEAD指向版本以及前一个版本之间code.cpp文件的不同

git diff HEAD HEAD^ -- code.cpp

# [git log 查看提交记录](https://www.cnblogs.com/xiaodi-js/p/7215365.html)

1. git log 查看提交历史记录

2. git log --oneline  或者 git log --pretty=oneline 以精简模式显示

3. git log --graph 以图形模式显示

4. git log --stat 显示文件更改列表

5. git log --author= 'name' 显示某个作者的日志

6. git log -p filepath 查看某个文件的详细修改

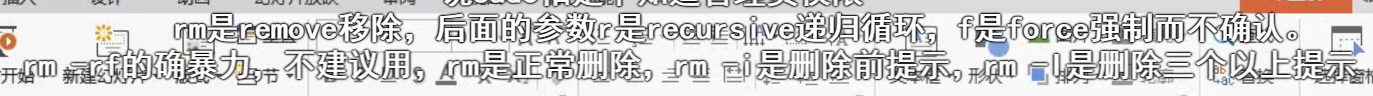
7. git log -L start,end:filepath 查看某个文件某几行范围内的修改记录

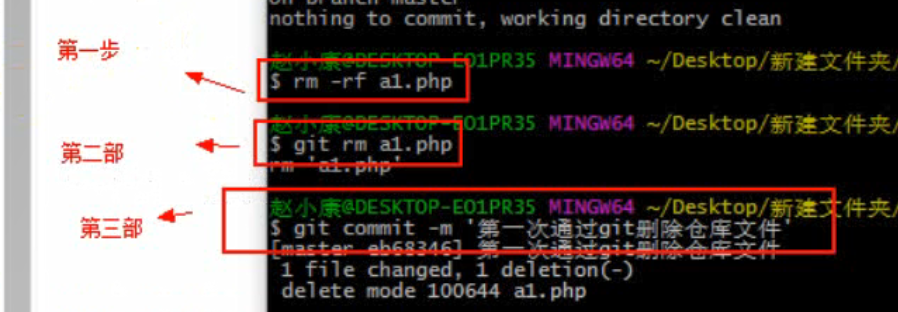
8. git log --stat commitId  或者 git show --stat commitId 查看某一次提交的文件修改列表

* 删除文件  
  1.使用rm删除工作区的一个文件，可以使其恢复

删除：rm -f code2.cpp

恢复：git checkout -- code2.cpp或者Git restore 文件名





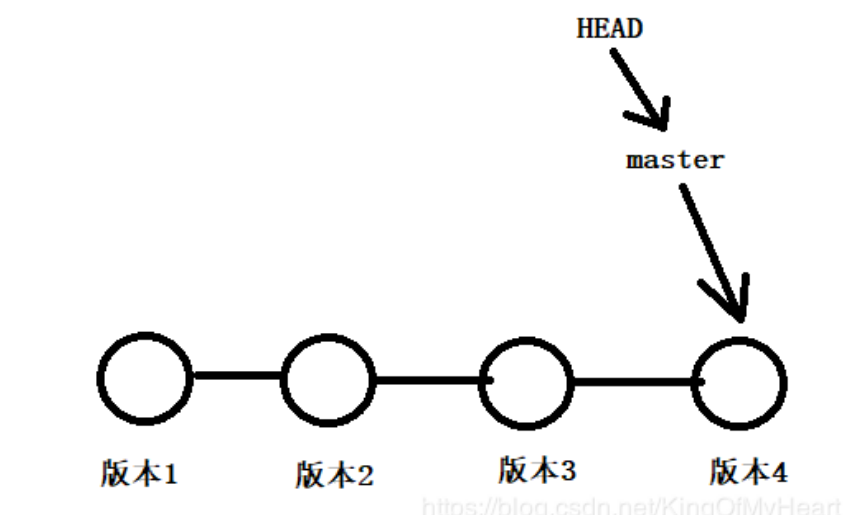
第一步删除本地文件，第二步删除暂存区文件。

2.确实要删除一个文件，将删除文件的改动放到暂存区中,然后创建新的版本记录

1. 删除文件：rm filename
2. 将删除文件操作记录在暂存区：git rm filename
3. 并且将删除文件的操作创建一个新的版本记录：git commit -m "删除了filename"

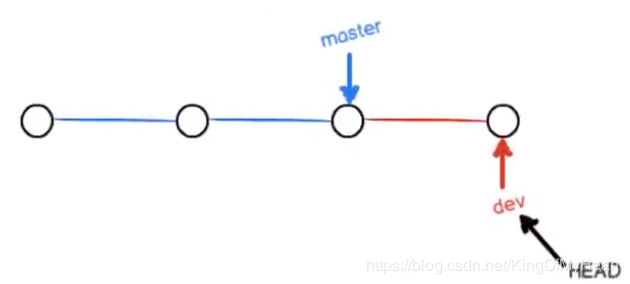
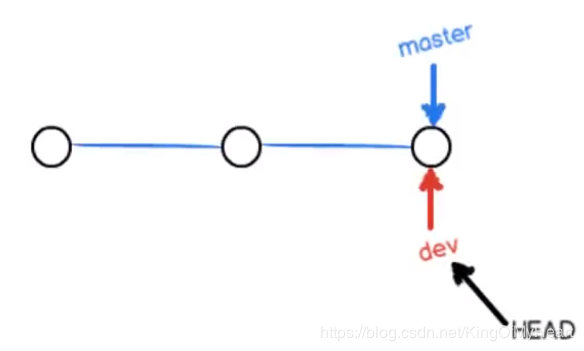
* 分支的分类与创建过程  
  **git把我们提交的多个版本串成了“一条线”，实际上这条线就是我们的分支；**

**1.master分支：**  
前面的所有git操作，都是在master分支上的，该分支也叫**主分支**。实际上，HEAD指向的是当前的分支，而分支指向的是我们当前的版本。



一开始，master是一条线，git使用master指向最新的提交版本，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点。

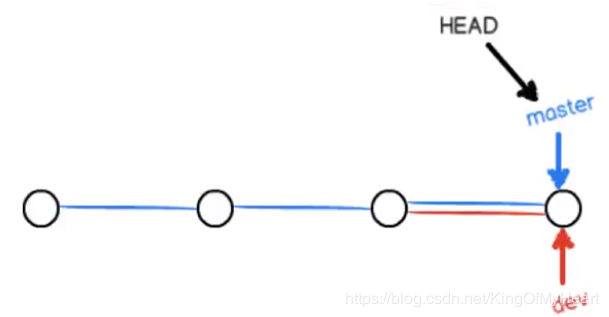
**2.创建新的分支的过程：**  
**2.1**假如我们要在master指向的位置开始创建一个新的分支，比如叫dev，此时git会创建一个新指针dev指向该位置，然后将HEAD指向dev，表示此时处于新分支dev上；



实际上这个速度比较快，只需要创建一个新指针，然后将HEAD指针指向该新指针，就表示了一个新分支的创建，工作区的文件并没有发生改变；

**2.2.** 接下来，我们的工作都是在dev分支上，而master指针一直在原来的位置上不动，我们每提交一个新版本，dev指针就向前挪动一下；

**2.3.** 假如我们在该分支上的工作完成了，那么就需要将dev合并到master分支上，git合并时最简单的方式是将master分支挪动到dev指向的位置



**2.4**. 如果需要删除dev分支时。直接将dev指针释放掉就好了

## **如何创建一个分支并切换**

1.查看当前在哪个分支下工作：git branch

2.创建一个分支并且切换到该分支下,比如创建一个dev分支。git checkout -b dev

（实际上，直接创建了一个指针指向了master指向的位置）（分支切换指的是，HEAD指向了我们新创建的分支）

3.在dev中修改之前的code.cpp文件并且创建新的版本。git add code.cpp git commit code.cpp

此时这些工作都是在dev分支下进行的，如下图；

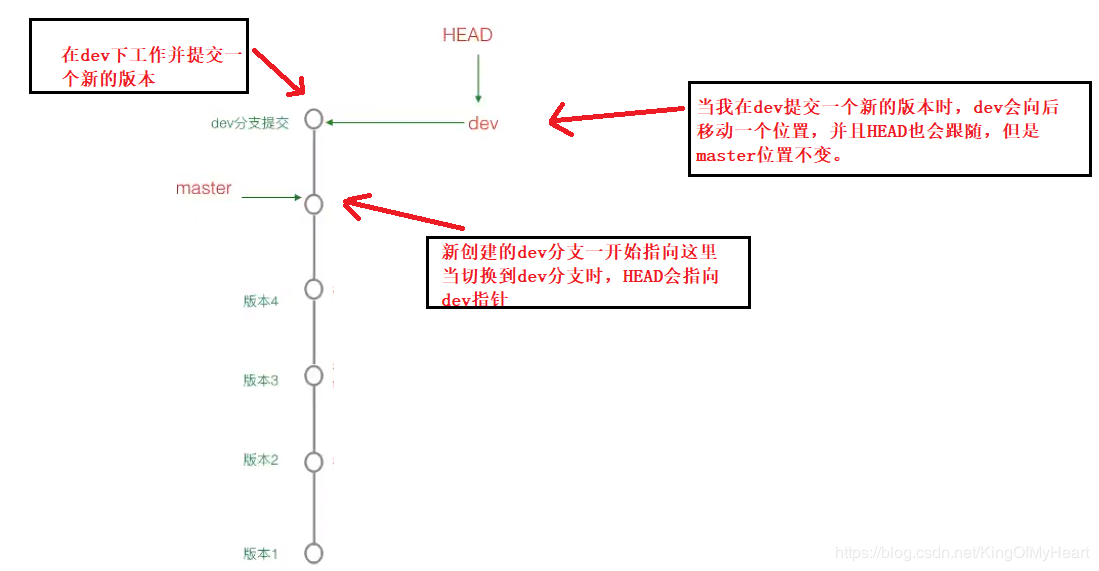
4.从dev分支切换到master分支。git checkout master(实际上，直接将HEAD指针指向了master)

5.合并分支，将dev合并到master。git merge dev (实际上是将master指向dev指向的位置) 这里使用的是快速合并，但是一些场合不适合该方式。

git rebase reking #合并分支reking和主干master，合并后的提交点属于主干master

6.删除分支，如删除dev分支。git branch -d dev

删除远程分支：git push origin --delete 分支名（直接写分支名）



**需要注意的是上面 合并分支的时候，使用的是快速合并方式，该方式不是所有模式下都适用。**  
**快速合并**：直接将master指针移动到dev指向的位置，然后dev分支的内容就属于了master了。

**命令总结：**

1. **查看分支：git branch**
2. **创建分支：git branch name**
3. **切换分支：git checkout name**
4. **创建并切换分支：git checkout -b name**
5. **合并分支到当前分支：git merge name**
6. **删除分支：git branch -d name**
7. $ git checkout origin/experimental切换到远程分支

# [看不到git远程分支](https://www.cnblogs.com/dashi/p/10146079.html)

1、先用fetch命令更新remote索引

$ git fetch

2、再查看remote分支，发现已经可以看到目标分支

$ git branch -a

3、再切换分支

$ git checkout dev-1.0

4、如需合并master分支到dev-1.0，则执行：

$ git merge master

## 创建分支及其相关命令

1. **查看分支**

git branch 查看本地仓库

git branch -r 查看远程仓库

1. **创建新分支**

//新建online分支

git checkout -b online

**3、查看当前所有分支**

//查看当前所有的分支

git branch -a

//结果显示带\*号的，而且颜色是绿色的即为我们当前所在的分支

\*master

Online

//查看远程所有分支

git branch -r # 查看远程所有分支

**4、切换分支**

//从当前的master分支切换到online分支上面git checkout online

**5、删除分支**

//删除online分支

git branch -d online

**6、本地分支重命名**

git branch -m <oldbranch> <newbranch> #重命名本地分支

# Git pull(拉取)，push(上传)命令整理

1、git拉取远程代码

git clone [https:*//xxx.git*](https://xxx.git)

2、git拉取远程指定分支下代码（-b  分支名称）

git clone -b v2.8.1 [https:*//xxx.git*](https://xxx.git)

3、初始化一个本地仓库，在同级目录下会出现一个隐藏的.git文件

git init

4、查看git的配置信息

git config -l

git config --list

5、本地提交到git索引（缓存）中

git add . #把新增的、修改的都加到缓存

git add -A #把新增、和修改的、和删除的都加到缓存

6、本地从缓存中加到文件库（repository）中

git commit -m '版本描述' #提交到文件库

7、本地一件提交到文件库中

git commit -am '版本描述' #一步完成提交

工作目录下面的所有文件都不外乎这两种状态：已跟踪(tracked)或未跟踪(untracked)。已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是未更新(unmodified)，已修改(modified)或者已放入暂存区(staged)

**使用git commit -am，则可以省略git add a.txt这一步，因为git commit -am可以提交跟踪过的文件，而a.txt一开始已经被跟踪过了,这两个命令的区别的关键就是git add命令**

8、git中的三类文件

被追踪的（tracked）:已经加入文档库

不被追踪的(untracked):没有加入文档库

忽略的(ignored):忽略那些不需要管理的文件夹或文件

9、查看本地添加了哪些远程分支地址

git remote

10、查看本地添加了哪些远程分支地址（详细）

git remote -v

11、删除本地指定的远程地址

git remote remove origin

12、添加远程地址

git remote add origin https:*//xxxxxxxxxxxx.git*

13、fetch与pull

1. fetch是将远程主机的最新内容拉到本地，不进行合并

git fetch origin master

1. pull 则是将远程主机的master分支最新内容拉下来后与当前本地分支直接合并 fetch+merge

git pull origin master

14、git分支说明

1. Production分支（主线分支用于发版，不会直接改）
2. Master分支，这个分支只能从其他分支合并，不能在这个分支直接修改
3. Develop分支（开发分支）
4. 这个分支是我们是我们的主开发分支，包含所有要发布到下一个Release的代码，这个主要合并与其他分支，比如Feature分支
5. Feature分支（新功能分支）
6. 这个分支主要是用来开发一个新的功能，一旦开发完成，我们合并回Develop分支进入下一个Release
7. Release分支（偏向测试）
8. 当你需要一个发布一个新Release的时候，我们基于Develop分支创建一个Release分支，完成Release后，我们合并到Master和Develop分支
9. Hotfix分支（紧急bug发布）
10. 当我们在Production发现新的Bug时候，我们需要创建一个Hotfix, 完成Hotfix后，我们合并回Master和Develop分支，所以Hotfix的改动会进入下一个Release

　　15、git中一些选项解释（重要）：

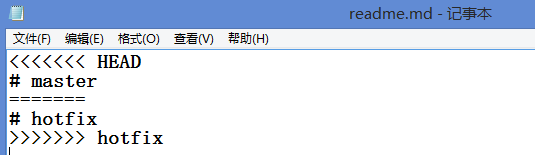
1. -d--delete：删除
2. -D--delete --force的快捷键
3. -f--force：强制
4. -m--move：移动或重命名
5. -M--move --force的快捷键
6. -r--remote：远程
7. -a--all：所有

# Git分支合并冲突处理

### 解决

（1）git status可以查看哪些文件有冲突没有合并，如图：



（2）打开readme.md文件，发现文件已经变成如下情况：

其中=======的上半部分对应的是master分支内容（HEAD指向当前分支，因为合并命令是在master分支中执行的），下半部分对应的是hotfix分支内容，现在就可以选择任何一个版本或者合并两个版本作为最终版本来解决冲突了。例如内容变为：

# master

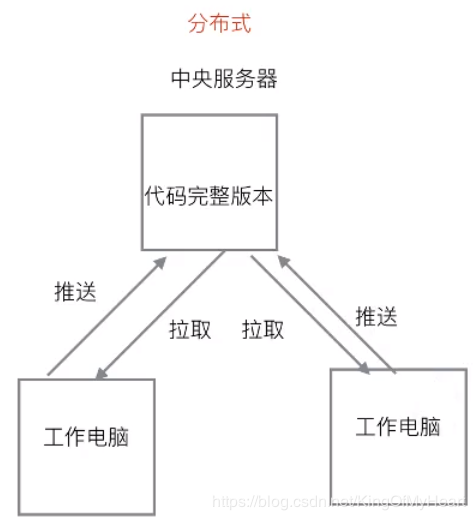
# hotfix

（3）冲突全部解决完后，在master分支下执行git add和git commit就可以完成最终版本提交，如图：

# Git(六)分支管理策略、如何禁用快速合并

<https://blog.csdn.net/KingOfMyHeart/article/details/89944352>

# Git(七)GitHub的使用、ssh的配置与使用流程



* add .gitignore:  
  我们希望git不要去给我们管理目录下的某种文件，比如忽略生成执行文件.o文件，就可以将\*.o写入.gitignore文件中，这样git就不会给我们管理这些文件，自动忽略，如果我们不忽略，git会一直提示我们没有被。
* 如何将一个项目克隆到我们自己的主机：
  + 1. 首先得拿到项目的地址，比如就拿我刚才创建的那个kingofmyheart项目来说

https://github.com/321919187/KingOfMyHeart

* + 1. 然后打开页面，点击clone拿到ssh协议下的地址，如图一

git@github.com:321919187/KingOfMyHeart.git

* + 1. 然后到我们主机的工作目录下：

git clone git@github.com:321919187/KingOfMyHeart.git

* 接下来就可以进行开发了，进行远程提交代码：
  + 1. 创建一个自己的分支，用来工作

git checkout -b king

* + 1. 然后就在这个分支上进行工作

coding......

* + 1. 推送我们的代码：

git push origin king

(将我们本地的分支king推送到远程，也就是推送到github，如果远程没有这个分支，会自动创建king分支)

* 如何用本地的分支 跟踪 远程(origin，也就是github )的分支：

设置用本地的king分支跟踪远程的king分支，就是将本地的分支和远程的分支关联起来：

git branch --set-upstream-to=origin/king king

(origin/king:远程的分支)

(king：本地的king分支)

#### 查看本地分支和远程分支的对应情况

git branch -vv 注意是两个v, 不是一个w!

#### 细节

对于github上的新仓库, 得先用git push -u origin master这种方式指定上游并提交一次后, 才能使用git branch --set-upstream-to=origin/master master

* 比如我在我本地的king分支中创建一个新的cpp文件，本地进行依次新的提交：

git add a.cpp

git commit -m "创建a.cpp文件"

然后通过git status 查看，会提示我们本地分支和远程的分支不一致，问我们需不需要push

只需要执行git push 即可

* 如何从远程的分支拉取代码：

git pull origin 分支名称

## 下载慢的问题：

C:\Windows\System32\drivers\etc

<https://blog.csdn.net/fanrenxiang/article/details/81508844?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task>

## 每次pull需要登录的问题

git config --global credential.helper store  
之后再次执行git push 或者git pull这时候还需要输入用户名和密码

下次就不需要了