Git笔记

一 、git 初始化

Git config user.name “jianhui”

Git config user.email [“1021807914@qq.com”](mailto:\“1021807914@qq.com\”)

Git add

Git status 查看仓库当前状态

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

二、Git add 各种区别

it add -A // 添加所有改动

git add \* // 添加新建文件和修改，但是不包括删除

git add . // 添加新建文件和修改，但是不包括删除

git add -u // 添加修改和删除，但是不包括新建文件

在 commit 前撤销 add:

gits

git reset <file> // 撤销提交单独文件

git reset // unstage all due changes

add/commit 前撤销对文件的修改:

git checkout -- README.md // 注意, add添加后(同commit提交后)就无法通过这种方式撤销修改

三、git 版本回退

git log命令显示从最近到最远的提交日志；

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交3628164...882e1e0（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

**Git reset -- hard HEAD^**

Git commit id号

Git reflog

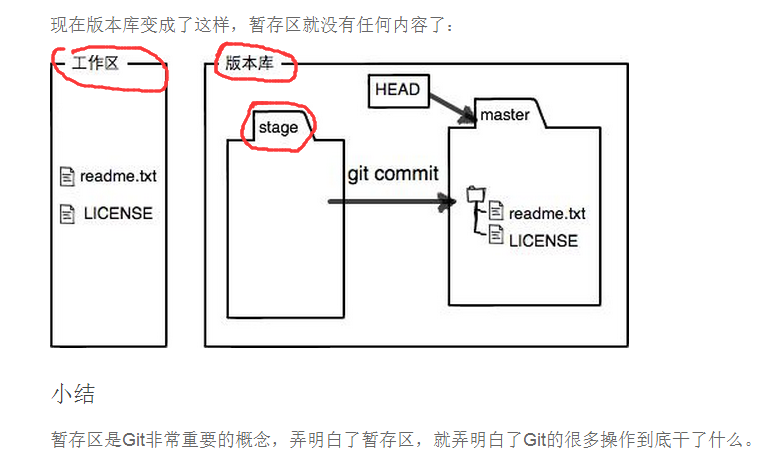
在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add distributed版本时，再想恢复到append GPL，就必须找到append GPL的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git **reset --hard commit\_id**。

穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

四、工作区和暂存区特别重要



五git管理的是修改 git diff 和 git diff --cached

git diff #是工作区(work dict)和暂存区(stage)的比较

git diff --cached #是暂存区(stage)和分支(master)的比较

Git管理的是修改，当你用git add命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

提交后，用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别：

总结：Git是如何跟踪修改的，每次修改，如果不add到暂存区，那就不会加入到commit中。

六、撤销修改 git checkout -- file 与 git reset HAED file

git checkout -- file可以丢弃工作区的修改：

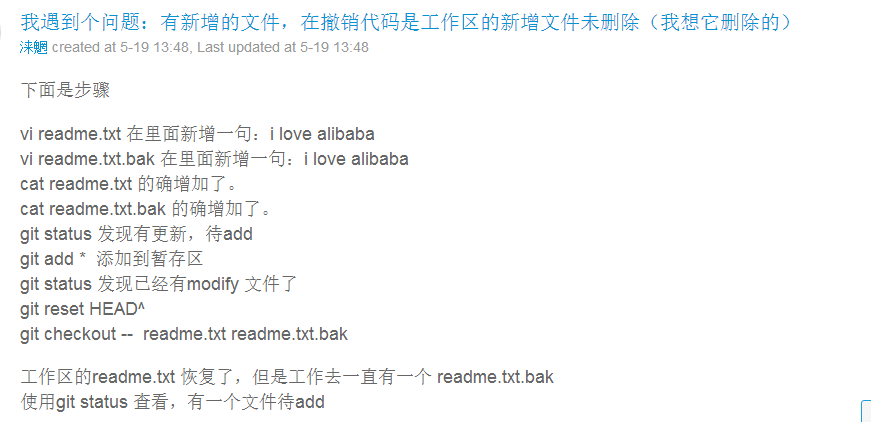


场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

**网友中问题：**



七、删除文件git rm

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：

$ git rm test.txt

rm 'test.txt'

$ git commit -m "remove test.txt"

[master d17efd8] remove test.txt

1 file changed, 1 deletion(-)

delete mode 100644 test.txt

现在，文件就从版本库中被删除了。

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

**$ git checkout -- test.txt**

git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

八、远程仓库 同过GitHub创建仓库

第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

$ ssh-keygen -t rsa -C "youremail@example.com"

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，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。

搭建GitHub仓库

http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/001374385852170d9c7adf13c30429b9660d0eb689dd43a000

九、添加远程库

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

十、Clone仓库

如果是window系统，将$ git clone git@github.com:liu9963xia/gitskills.git 改成 $ git clone https://github.com/liu9963xia/gitskills.git 就行了

十一、创建余合并分支

Git checkout -b dev 创建新分支

查看分支：git branch

创建分支：git branch <name>

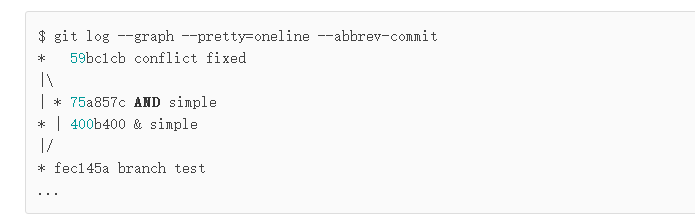
切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

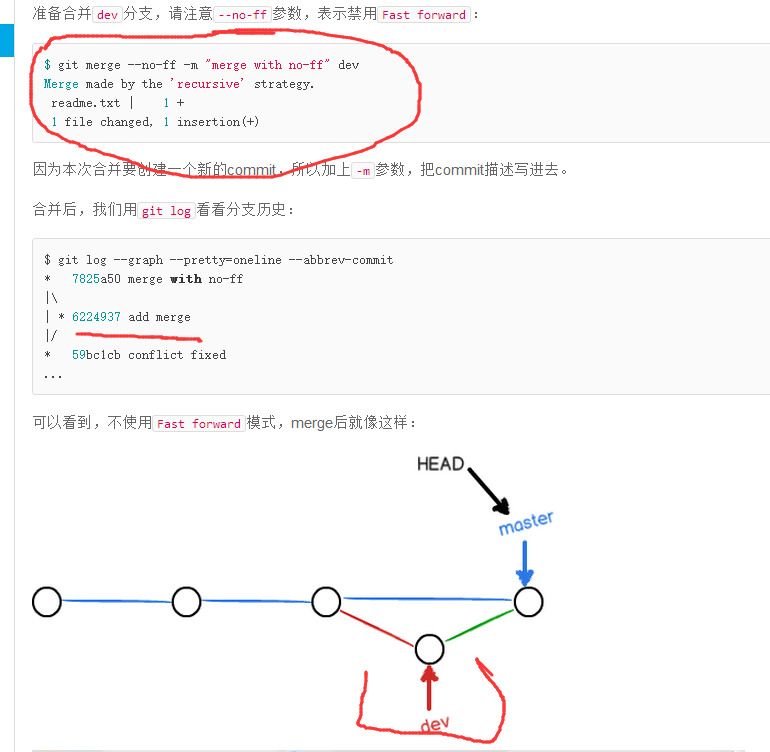
合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

Gitlog 查看分支的合并情况



十二、分支管理策略 --no-ff 参数



十三、Bug分支 git stash 储藏功能

Git stash list

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：

一是用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；

另一种方式是用git stash pop，恢复的同时把stash内容也删了：

$ git stash apply stash@{0}

十四、feature分支

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

多人协作

Git remote -v 查看远程库的信息

因此，多人协作的工作模式通常是这样：

首先，可以试图用git push origin branch-name推送自己的修改；

如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用git push origin branch-name推送就能成功！

如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没有创建，

用命令git branch --set-upstream branch-name origin/branch-name。

小结

查看远程库信息，使用git remote -v；

本地新建的分支如果不推送到远程，对其他人就是不可见的；

从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；

在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

建立本地分支和远程分支的关联，

使用git branch --set-upstream branch-name origin/branch-name；

从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

十五、标签管理

**[Git](http://lib.csdn.net/base/git" \o "Git知识库" \t "http://blog.csdn.net/rainnings/article/details/_blank)** 中的tag指向一次commit的id，通常用来给开发分支做一个标记，如标记一个版本号。

**打标签**

git tag -a v1.01 -m "Relase version 1.01"

注解：git tag 是打标签的命令，-a 是添加标签，其后要跟新标签号，-m 及后面的字符串是对该标签的注释。

**提交标签到远程仓库**  
git push origin -tags  
注解：就像git push origin master 把本地修改提交到远程仓库一样，-tags可以把本地的打的标签全部提交到远程仓库。  
**删除标签**  
git tag -d v1.01  
注解：-d 表示删除，后面跟要删除的tag名字  
**删除远程标签**  
git push origin :refs/tags/v1.01  
注解：就像git push origin :branch\_1 可以删除远程仓库的分支branch\_1一样， 冒号前为空表示删除远程仓库的tag。  
**查看标签**  
git tag或者

git tag -l

1查看tag

列出所有tag

git tag

这样列出的tag是按字母排序的，和创建时间没关系。如果只是想查看某些tag的话，可以加限定：

git tag -l version1.\*

这样就只会列出1.几的版本。

2创建tag

创建轻量级tag：

git tag version 1.0 也可以 git tag 1.0

带有信息的tag

git tag -a version1.0 -m 'first version'

-m 后面带的就是注释信息，一般写当前的版本作用，这种是普通tag

签名的tag：

git tag -s version1.0 -m 'first version'

前提是你有GPG私钥，把上面的a换成s就行了。除了可以为当前的进度添加tag，还可以为以前的commit添加tag：

首先查看以前的com

删除标签tag

小结

命令git push origin <tagname>可以推送一个本地标签；

命令git push origin --tags可以推送全部未推送过的本地标签；

命令git tag -d <tagname>可以删除一个本地标签；

命令git push origin :refs/tags/<tagname>可以删除一个远程标签。

十六、GitHub上fork开源项目

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。

十七、忽略特殊文件

忽略某些文件时，需要编写.gitignore；

.gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理！

案例



十八、配置别名 参考廖雪峰的blog

git config --global **alias**.st status

git config --global **alias**.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"

十九 git clone /remote fetch pull push 的用法



## 一、git clone

远程操作的第一步，通常是从远程主机克隆一个版本库，这时就要用到git clone命令。

$ git clone <版本库的网址>

比如，克隆jQuery的版本库。

$ git clone https://github.com/jquery/jquery.git

该命令会在本地主机生成一个目录，与远程主机的版本库同名。如果要指定不同的目录名，可以将目录名作为git clone命令的第二个参数。

$ git clone <版本库的网址> <本地目录名>

git clone支持多种协议，除了HTTP(s)以外，还支持SSH、Git、本地文件协议等，下面是一些例子。

$ git clone http[s]://example.com/path/to/repo.git/

$ git clone ssh://example.com/path/to/repo.git/

$ git clone git://example.com/path/to/repo.git/

$ git clone /opt/git/project.git

$ git clone file:///opt/git/project.git$ git clone ftp[s]://example.com/path/to/repo.git/

$ git clone rsync://example.com/path/to/repo.git/

SSH协议还有另一种写法。

$ git clone [user@]example.com:path/to/repo.git/

通常来说，Git协议下载速度最快，SSH协议用于需要用户认证的场合。各种协议优劣的详细讨论请参考[官方文档](http://git-scm.com/book/en/Git-on-the-Server-The-Protocols" \t "http://blog.csdn.net/u012575819/article/details/_blank)。

## 二、git remote

为了便于管理，Git要求每个远程主机都必须指定一个主机名。git remote命令就用于管理主机名。

不带选项的时候，git remote命令列出所有远程主机。

$ git remote

origin

使用-v选项，可以参看远程主机的网址。

$ git remote -v

origin [git@github](mailto:git@github" \t "http://blog.csdn.net/u012575819/article/details/_blank).com:jquery/jquery.git (fetch)

origin [git@github](mailto:git@github" \t "http://blog.csdn.net/u012575819/article/details/_blank).com:jquery/jquery.git (push)

上面命令表示，当前只有一台远程主机，叫做origin，以及它的网址。

克隆版本库的时候，所使用的远程主机自动被Git命名为origin。如果想用其他的主机名，需要用git clone命令的-o选项指定。

$ git clone -o jQuery https://github.com/jquery/jquery.git

$ git remote

jQuery

上面命令表示，克隆的时候，指定远程主机叫做jQuery。

git remote show命令加上主机名，可以查看该主机的详细信息。

$ git remote show <主机名>

git remote add命令用于添加远程主机。

$ git remote add <主机名> <网址>

git remote rm命令用于删除远程主机。

$ git remote rm <主机名>

git remote rename命令用于远程主机的改名。

$ git remote rename <原主机名> <新主机名>

## 三、git fetch

一旦远程主机的版本库有了更新（Git术语叫做commit），需要将这些更新取回本地，这时就要用到git fetch命令。

$ git fetch <远程主机名>

上面命令将某个远程主机的更新，全部取回本地。

git fetch命令通常用来查看其他人的进程，因为它取回的代码对你本地的开发代码没有影响。

默认情况下，git fetch取回所有分支（branch）的更新。如果只想取回特定分支的更新，可以指定分支名。

$ git fetch <远程主机名> <分支名>

比如，取回origin主机的master分支。

$ git fetch origin master

所取回的更新，在本地主机上要用"远程主机名/分支名"的形式读取。比如origin主机的master，就要用origin/master读取。

git branch命令的-r选项，可以用来查看远程分支，-a选项查看所有分支。

$ git branch -r

origin/master

$ git branch -a\* master

remotes/origin/master

上面命令表示，本地主机的当前分支是master，远程分支是origin/master。

取回远程主机的更新以后，可以在它的基础上，使用git checkout命令创建一个新的分支。

$ git checkout -b newBrach origin/master

上面命令表示，在origin/master的基础上，创建一个新分支。

此外，也可以使用git merge命令或者git rebase命令，在本地分支上合并远程分支。

$ git merge origin/master

# 或者

$ git rebase origin/master

上面命令表示在当前分支上，合并origin/master。

## 四、git pull

git pull命令的作用是，取回远程主机某个分支的更新，再与本地的指定分支合并。它的完整格式稍稍有点复杂。

$ git pull <远程主机名> <远程分支名>:<本地分支名>

比如，取回origin主机的next分支，与本地的master分支合并，需要写成下面这样。

$ git pull origin next:master

如果远程分支是与当前分支合并，则冒号后面的部分可以省略。

$ git pull origin next

上面命令表示，取回origin/next分支，再与当前分支合并。实质上，这等同于先做git fetch，再做git merge。

$ git fetch origin

$ git merge origin/next

在某些场合，Git会自动在本地分支与远程分支之间，建立一种追踪关系（tracking）。比如，在git clone的时候，所有本地分支默认与远程主机的同名分支，建立追踪关系，也就是说，本地的master分支自动"追踪"origin/master分支。

Git也允许手动建立追踪关系

git branch --set-upstream master origin/next

上面命令指定master分支追踪origin/next分支。

如果当前分支与远程分支存在追踪关系，git pull就可以省略远程分支名。

$ git pull origin

上面命令表示，本地的当前分支自动与对应的origin主机"追踪分支"（remote-tracking branch）进行合并。

如果当前分支只有一个追踪分支，连远程主机名都可以省略。

$ git pull

上面命令表示，当前分支自动与唯一一个追踪分支进行合并。

如果合并需要采用rebase模式，可以使用--rebase选项。

$ git pull --rebase <远程主机名> <远程分支名>:<本地分支名>

如果远程主机删除了某个分支，默认情况下，git pull 不会在拉取远程分支的时候，删除对应的本地分支。这是为了防止，由于其他人操作了远程主机，导致git pull不知不觉删除了本地分支。

但是，你可以改变这个行为，加上参数 -p 就会在本地删除远程已经删除的分支。

$ git pull -p

# 等同于下面的命令

$ git fetch --prune origin

$ git fetch -p

## 五、git push

git push命令用于将本地分支的更新，推送到远程主机。它的格式与git pull命令相仿。

$ git push <远程主机名> <本地分支名>:<远程分支名>

注意，分支推送顺序的写法是<来源地>:<目的地>，所以git pull是<远程分支>:<本地分支>，而git push是<本地分支>:<远程分支>。

如果省略远程分支名，则表示将本地分支推送与之存在"追踪关系"的远程分支（通常两者同名），如果该远程分支不存在，则会被新建。

$ git push origin master

上面命令表示，将本地的master分支推送到origin主机的master分支。如果后者不存在，则会被新建。

如果省略本地分支名，则表示删除指定的远程分支，因为这等同于推送一个空的本地分支到远程分支。

$ git push origin :master

# 等同于

$ git push origin --delete master

上面命令表示删除origin主机的master分支。

如果当前分支与远程分支之间存在追踪关系，则本地分支和远程分支都可以省略。

$ git push origin

上面命令表示，将当前分支推送到origin主机的对应分支。

如果当前分支只有一个追踪分支，那么主机名都可以省略。

$ git push

如果当前分支与多个主机存在追踪关系，则可以使用-u选项指定一个默认主机，这样后面就可以不加任何参数使用git push。

$ git push -u origin master

上面命令将本地的master分支推送到origin主机，同时指定origin为默认主机，后面就可以不加任何参数使用git push了。

不带任何参数的git push，默认只推送当前分支，这叫做simple方式。此外，还有一种matching方式，会推送所有有对应的远程分支的本地分支。Git 2.0版本之前，默认采用matching方法，现在改为默认采用simple方式。如果要修改这个设置，可以采用git config命令。

$ git config --global push.default matching

# 或者

$ git config --global push.default simple

还有一种情况，就是不管是否存在对应的远程分支，将本地的所有分支都推送到远程主机，这时需要使用--all选项。

$ git push --all origin

上面命令表示，将所有本地分支都推送到origin主机。

如果远程主机的版本比本地版本更新，推送时Git会报错，要求先在本地做git pull合并差异，然后再推送到远程主机。这时，如果你一定要推送，可以使用--force选项。

$ git push --force origin

上面命令使用--force选项，结果导致远程主机上更新的版本被覆盖。除非你很确定要这样做，否则应该尽量避免使用--force选项。

最后，git push不会推送标签（tag），除非使用--tags选项。

$ git push origin --tags



服务器名:双线一区（点卡区）比翼齐飞（点卡服）；

角色名:范冰冰姐

密码: jk5926265. (注意后面有个点)

职业：万花

现等级：1

手机：18075876985