**常用的git命令**

**--文件目录操作命令**

1 mkdir \* 创建一个空目录 \*指目录名

2 pwd 显示当前目录的路径。

3 cat \* 查看\*文件内容

4 git rm \* 删除\*\*文件

**--git初始化操作**

1 git init 把当前的目录变成git仓库，生成隐藏.git文件。

2 git remote add origin url 把本地仓库的内容推送到GitHub仓库。

3 git clone git@url/test.git 从远程库克隆

4 git add \* 把x文件添加到暂存区去。

5 git commit –m "\*" 提交文件 –m 后面的是注释。

**--git 克隆分支**

1 git clone xxx.git 最简单直接的命令

2 git clone xxx.git "指定目录" clone到指定目录

3 git clone -b branchname xxx.git clone时创建新的分支替代默认Origin HEAD（master）

**--clone 远程分支**

**git clone 命令默认的只会建立master分支**，如果你想clone指定的某一远程分支(如：dev)的话，可以如下：

　　1. 查看所有分支(包括隐藏的)  git branch -a 显示所有分支

1 \* master

2 remotes/origin/HEAD -> origin/master

3 remotes/origin/dev

4 remotes/origin/master

　　2. **在本地新建同名的("dev")分支，并切换到该分支**

1 git checkout -t origin/dev 该命令等同于：

2 git checkout -b dev origin/dev

**--查看命令**

1 git status 查看仓库状态

2 git diff \* 查看X文件修改了那些内容

3 git log 查看历史记录

4 git reflog 查看历史记录的版本号id（记录你的每一次命令,不论是否提交）

5 git log --pretty=oneline 如果信息量太多可以进行比较好的列表显示

**--版本回退**

1 git reset –hard HEAD^ 回退到上一个版本

2 git reset --hard HEAD~第几个 如果想回退到第3个版本，使用git reset –hard HEAD~3

3 git reset --hard 057d 回退到某一个具体的版本号

**--撤销修改**

1 git checkout file-name 恢复某个已修改的文件（撤销未提交的修改）：

2 git revert HEAD 还原最近一次提交的修改：

3 git revert commit-id 还原指定版本的修改

**--分支管理**

复制代码

1 git branch 查看本地所有的分支

2 git branch -a 查看远程所有的分支

3 git branch name 创建分支

4 git branch –d dev 删除dev分支

5 git push origin --delete dev 删除远程的dev分支

6 git branch -m dev develop 重命名分支

7 git checkout –b dev 创建dev分支 并切换到dev分支上

8 git merge dev 在当前分支上合并dev分支代

9 git push origin zyf-dev 把当前新疆的zyf-dev分支推送到远程库(远程仓库没有给分支则会新建立该分支)

10 git checkout — \* 把XX文件在工作区的修改全部撤销。

11 git checkout master 切换回master分支

12 git push --set-upstream origin dev 提交修改并创建远程分支dev

复制代码

**--tag相关操作**

1 git tag 列出所有的tag

2 git tag name 打轻量标签 name

3 git tag -d name 删除本地的tag

4 git push origin --delete tag name 删除远程的tag

5 git show name 查看tag信息

6 git push origin name 将tag提交到远程

**--隐藏的文件**

1 git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作

2 git stash list 查看所有被隐藏的文件列表

3 git stash apply 恢复被隐藏的文件，但是内容不删除

4 git stash drop 删除文件

5 git stash pop 恢复文件的同时 也删除文件

**--查看远程库信息(git  remote的用法)**

1 git remote 查看远程库的信息

2 git remote –v 查看远程库的详细信息

3 git remote add name url 添加远程仓库

4 git remote rename oldname newname 重命名仓库

5 git remote rm 删除仓库

**--将远程分支拉取到本地**

1 方法一：git checkout -b 本地分支名x origin/远程分支名x

2 方式二：git fetch origin 远程分支名x:本地分支名x

**--git pull操作**

复制代码

1 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，基本的格式如下。

2 $ git pull <远程主机名> <远程分支名>:<本地分支名>

3

4 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

5 $ git pull origin next:master

6

7 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。

8 $ git pull origin next

9

10 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。

11 $ git fetch origin

12 $ git merge origin/next

13

14 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。

15 Git也允许手动建立追踪关系。

16 git branch --set-upstream master origin/next

17

18 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。

19 $ git pull origin

复制代码

**--git 设置大小写敏感**

Windows上的Git默认是大小写不敏感的，这样多平台写作就可能会出现问题。Win上的Git设置为大小写敏感的命令如下

1 git config core.ignorecase false

**--git 设置忽略文件或文件夹权限修改**

1 git config core.filemode false

**--创建追踪分支**

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 (最好使用这种方式)

  $ git branch --track  master origin/master //在使用 git branch 命令时加上 '--track' 参数, 来手动创建一个追踪分支

**--切换git 命令提示中文到英文**

// ubuntu装的git不知道怎么就出现全中文的提示,不太好,果断切换到中文了,切换方法如下:  
// 1:写入

echo "alias git='LANG=en\_GB git'" >> ~/.bashrc

// 2:生效

source ~/.bashrc

**--git 删除未添加到版本中的文件或者文件夹**

git checkout 只能回退在版本中的修改或者删除, 对于新添加的文件是没有作用的, 也就是说, 新建的文件或者文件夹是:Untracked files, 要删除或者清理掉这些文件,需要使用 git clean 命令:

复制代码

1 // 删除 untracked files

2 git clean -f

3

4 // 连 untracked 的目录也一起删掉

5 git clean -fd

6

7 // 连 gitignore的untrack 文件/目录也一起删掉 （一般这个是用来删掉编译出来的 .o一类的文件）

8 git clean -xfd

9

10 // 在使用清理 git clean之前，建议加上 -n 来先看看会删掉哪些文件，防止重要文件被误删

11 git clean -nxfd

12 git clean -nf

13 git clean -nfd

复制代码

**三、常出错误：**  
**1：There is no tracking information for the current branch...**  
    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。  
  
**2: ![rejected] dev -> dev (non-fast-forward)  ... Updates were rejected because the tip of your current branch**  
    推送失败，因为远程代码的最新提交和你试图推送的提交有冲突，解决办法也很简单，先用git pull把最新的提交从origin/dev抓下来，然后，在本地合并，解决冲突，再推送  
      
**3：CONFLICT (content): Merge conflict in . . .**  
    这回git pull成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的解决冲突一样。解决后，提交，再push：  
      
**4: You are not currently on a branch, so I cannot use any**  
   解决办法：git checkout master  
  
**5：Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.**  
   上面的显示的意思是，有一个更新还没有反应到本地来，可能是别人往server上提交了一点东西。   
   可以使用git pull命令拿这些更新到本地来。

**6：在执行 git push 时可能会看到如下消息:**

warning: push.default is unset; its implicit value is changing in

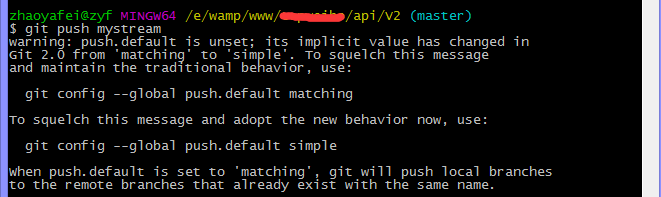
Git 2.0 from 'matching' to 'simple'. To squelch this message

and maintain the current behavior after the default changes, use:

git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

git config --global push.default simpl



  解决办法：‘matching’ 参数是 Git 1.x 的默认行为，如果你执行 git push 但没有指定分支，它将 push 所有你本地的分支到远程仓库中对应匹配的分支。而 Git 2.x 默认的是 simple，意味着执行 git push 没有指定分支时，只有当前分支会被 push 到你使用 git pull 获取的代码。   
根据提示，修改git push:

1 git config --global push.default matching

再次执行git push 就行了。

**7：You asked to pull from the remote 'origin', but did not specify:a branch. Because this is not the default configured remotefor your current branch, you must specify a branch on the command line.**

解决办法：找到：.git/config  修改如下

1 [branch "master"]

2 remote = origin

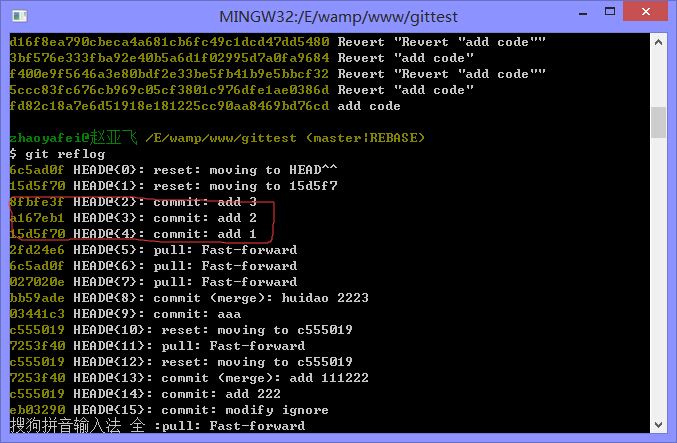
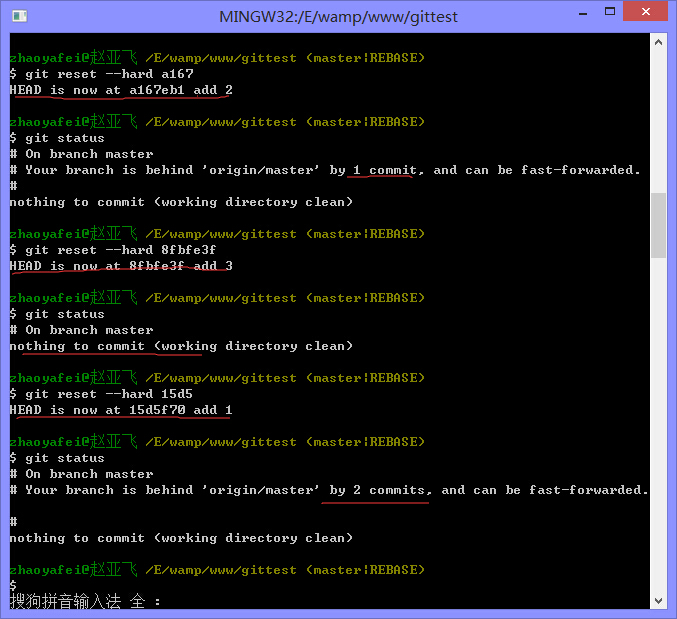
3 merge = refs/heads/master

**8：ERROR: Permission to user1/test.git denied to user2   fatal: The remote end hung up unexpectedly**

　账户冲突，下面有针对讲解

**9: 添加的ssh不起作用？**

   ssh 的添加一定要在root用户权限在添加，其他的权限不起作用，切记！

**四、经验之谈：**  
1：git revert 和 git reset的区别   
开始感觉两者一样，但是在客户端和命令行窗口操作之后发现大大的不同，看一个例子：我先提交三次1,2,3 id号分别是（15d5f70,a167eb1,8fbfe3f）如下图  
  
然后执行版本回退 git reset –hard a167 如图：  


说明执行git reset –hard \*\* 版本回退是撤销某次提交，但是此次之后的修改都会被退回到暂存区，因为上面显示有提交（by 1 commit）  
同理执行 git revert –hard a167  
git revert是用一次新的commit来回滚之前的commit  
　　在回滚操作上看，效果一样。但是在日后继续merge以前的老版本时有区别。revert是用一次逆向的commit“中和”之前的提交，因此日后合并老的branch时，导致这部分改变不会再次出现，但是reset是之间把某些commit在某个branch上删除，因而和老的branch再次merge时，这些被回滚的commit应该还会被引入。   
得到结论：git reset 是把HEAD向后移动了一下，而git revert是HEAD继续前进，  
只是新的commit的内容和要revert的内容正好相反，能够抵消要被revert的内容。

　　这样在版本回退的时候就比较的好，比如本地dev分支，远程master分支，某一次的上线发现线上有重大的bug，然后你reset本地代码到前一个版本，在你git push 的时候git会报一个错误：“说你的本地分支版本低于远程master分支的版本，让你先去pull 下来代码，再提交”，这就不行了，你pull下来的代码还是最新版有bug的代码，不进行pull代码，master分支又不让你提交。解决这个问题还得使用 revert 来进行 “回退”操作，为什么加引号呢，是因为这种回退是向前提交一次中和了上次的修改，这就比较好了，这样你revert之后相当于指针向前移动一次，本地版本dev高于远程master版本，这时你就可以git push 本地代码到远程了。也就是说，git reset 对未提交到（git push）远程的修改做回滚比较好，如果要回滚本地，同时想远程也回滚，就要用到revert  
  
**五、常用修改：**  
**1：避免每次提交都输入用户名和密码**  
    原因是使用了https方式 puh 在termail里边 输入  git remote -v 可以看到形如一下的返回结果  
    origin https://github.com/git/librarysystem.git(fetch)  
    origin https://github.com/git/librarysystem.git(push)  
    下面把它换成ssh方式的。

1. git remote rm origin

2. git remote add origin git@github.com:yu/demo.git

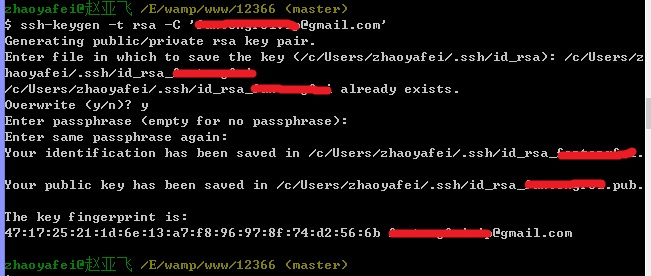
3. git push origin

**2：添加忽略文件 .gitignore**  
    .gitignore配置文件用于配置不需要加入版本管理的文件。  
    配置语法： "/"  开头表示目录；  "\*"  通配多个字符；  "?"  通配单个字符   "[]" 包含单个字符的匹配列表；  "!"  表示不忽略(跟踪)匹配到的文件或目录；  
    说明：git 对于 .ignore 配置文件是按行从上到下进行规则匹配的，意味着如果前面的规则匹配的范围更大，则后面的规则将不会生效；  
    示例：（1） foder/\*     忽略目录foder下的全部内容,不管是根目录下的 /foder/ 目录，还是某个子目录 /list/foder/ 目录，都会被忽略。  
    　　   （2） /foder/\*   说明：忽略根目录下的 /foder/ 目录的全部内容；  
    　　   （3） /\*!.gitignore  
                          !/fw/bin/  
                          !/fw/sf/  
                          说明：忽略全部内容，但是不忽略 .gitignore文件,根目录下的 /fw/bin/ 和 /fw/sf/   
   注意问题： .gitignore文件只对还没有加入版本管理的文件起作用，如果之前已经用git把要忽略的文件纳入了版本库，就不起作用了。

**3：如何在同一台电脑上使用两个git账户**

　　问题描述：作为程序员有时候是有多个的项目，而且每个项目git的用户名，邮箱等，都不一样。就我来说，我在github上有一个账号，自己开发一些东西，同时我参与的也有项目需要连接别人的github账号，在我电脑上只有一个git客户端，本地一个身份用户，开始的时候我自己开发是用ssh方式，参与的项目的是用https连接，但是https每次都需要输入密码，非常不方便，所以想转成ssh连接，利用命令：  
>$ git remote rm origin  
>$ git remote add origin git@github.com:yu/demo.git   
>$ git push origin         
转换成功，不幸的是push的时候出错了：ERROR: Permission to user1 denied to  
这种情况的解决方式是方法，使用本地托管多个个ssh的密钥，不同的账号是用不同的密匙。  
**第一步：生成SSH Key**

执行命令，如下图：



具体说明：

复制代码

1 $ ssh-keygen -t rsa -C "your\_email@youremail.com"

2 Creates a new ssh key using the provided email Generating public/private rsa key pair.

3 #输入将要保存的路径，我的默认为当前路径（/c/Users/zhaoyafei/.ssh/id\_rsa）,但是不能使用默认文件id\_rsa，要添加新账户就要生起一个成新的好记的名字，例如id\_rsa\_work

4 Enter file in which to save the key (/c/Users/zhaoyafei/.ssh/id\_rsa):

5 #输入回车后提示输入一个类似于密码的号，直接回车就行

6 Enter passphrase (empty for no passphrase):

7 Enter same passphrase again:

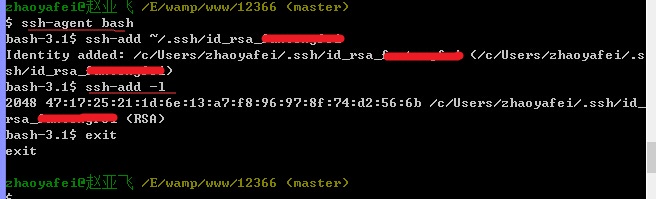
复制代码

**第二步：识别新的私钥**  
    默认SSH只会读取id\_rsa，所以为了让SSH识别新的私钥，需要将其添加到SSH agent

1 ssh-add ～/.ssh/id\_rsa\_work

2 //该命令如果报错：Could not open a connection to your authentication agent.可执行 ssh-agent bash 命令后再执行ssh-add -l 看是否成功。

执行命令，如下图：



**第三步：修改config文件**  
    该文件用于配置私钥对应的服务器。内容如下:

复制代码

1 # Default github （默认的）

2 Host github.com

3 HostName github.com

4 User git

5 IdentityFile ~/.ssh/id\_rsa

6

7 # second user(work@gmail.com)

8 Host github\_work

9 HostName github.com

10 User git

11 IdentityFile ~/.ssh/id\_rsa\_work

复制代码

       这样配置，也就是使用hostname为github.com会根据用户名的不同，去使用不用的private key。github上，也可以添加对应的公钥。其实这个配置是关于ssh的与git无关，只是git使用的方式是ssh的方式。Host可随意，方便自己记忆，后续在添加remote是还需要用到。

**第四步：将SSH key输入到GitHub网站中**      
      将生成的id\_rsa\_work.pub输入到GitHub网站中就可以了，输入完成后单击Add key后，会看到git-tutorial已经被添加进去了。 配置完成后，在连接非默认帐号的github仓库时，远程库的地址要对应地做一些修改，比如现在添加abc帐号下的一个仓库test，则需要这样添加：git remote add test git@github-work:abc/test.git #并非原来的git@github.com:abc/test.git 这样每次连接都会使用id\_rsa\_abc与服务器进行连接。

　　**注意：**github根据配置文件的user.email来获取github帐号显示author信息，所以对于多帐号用户一定要记得将user.email改为相应的email(work@mail.com)。我遇到的问题是本地单一用户，链接不同的github账号，所以用户配置可以采用全局配置。如果不同本地用户（不同的身份标识，即需要本地使用不同的用户名），那么git就不要使用有–global的配置，可以按照下面的方法：

1 1)设置局部的user.name和user.email

2 git config user.name "xxxxxx"

3 git config user.email "xxx@xxx.com"

4 2)设置全局的user.name和user.email

5 git config --global user.name "xxxxxx"

6 git config –-global user.email ""

**六、题外话（相关概念）**

**Git - 版本控制工具**  
　　Git是一个开源的分布式版本控制系统，用以有效、高速的处理从很小到非常大的项目版本管理。Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。Torvalds 开始着手开发 Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得 BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。例如 最近就迁移到 Git 上来了，很多 Freedesktop 的项目也迁移到了 Git 上。

**Github - 一个网站，提供给用户空间创建git仓储，保存用户的一些数据文档或者代码等**  
　　作为开源代码库以及版本控制系统，Github目前拥有140多万开发者用户。随着越来越多的应用程序转移到了云上，Github已经成为了管理软件开发以及发现已有代码的首选方法。如前所述，作为一个分布式的版本控制系统，在Git中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并。GitHub可以托管各种git库，并提供一个web界面，但与其它像 SourceForge或Google Code这样的服务不同，GitHub的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的“fork”的按 钮，然后将代码检出并将修改加入到刚才分出的代码库中，最后通过内建的“pull request”机制向项目负责人申请代码合并。已经有人将GitHub称为代码玩家的MySpace。

**GitLab - 基于Git的项目管理软件**  
GitLab 是一个用于仓库管理系统的开源项目。使用Git作为代码管理工具，并在此基础上搭建起来的web服务。三者都是基于git的，可以说是git的衍生品。