

**我想合并现在的提交到上一次提交，怎么做？**

A：用git commit --amend命令

**怎么忽略某些文件，让它们不在版本控制之中？**

A：在.gitignore文件中描述这些文件

**Q：操作系统中移除文件和git rm命令有什么区别？**

A：操作系统移除，文件快照可能还在暂存区和仓库区；git rm不仅在系统里面删掉了文件，还将其快照移出了暂存区和仓库区。

**Q：只移除快照，但保留文件，怎么做？**

A：git rm命令带上--cached命令

**Q：如何查看提交日志，怎么把每次提交信息在日志中浓缩一行？**

A：git log命令可查看日志，带上--pretty=oneline可以把每次提交信息展示为一行

**Q：如何对比工作区文件、暂存区、仓库区快照的具体差别，如何对比两次提交之间的差异？**

A：git diff命令可以对比工作区-暂存区，带上--cached命令，对比暂存区-仓库区，带上commit\_id对比工作区和仓库区，带上两个commit\_id对比两次提交之间的差异。

<http://blog.jobbole.com/78960/> （推荐好文）

<http://mp.weixin.qq.com/s?__biz=MzA4ODIwMDUyNw==&mid=2652988644&idx=1&sn=64fce80ee7ab7186be6527d12ac74e63&scene=1&srcid=0906er4ecgKwrFsdutwSS4fP#rd>

<http://blog.csdn.net/renfufei/article/details/37725057/>

<http://dztro.blog.163.com/blog/static/2366730252014102732814915/>

# 基本概念，注册Github

-----------基本概念，注册Github------------

**仓库（Respository）:**存放项目代码，一个项目对应一个仓库

**收藏（Star）：**收藏项目，方便下次查看

**复制克隆项目（Fork）:**fored form 项目来源，独立存在；在原有项目的基础上新建一个分支，改动不会影响原项目

**发起请求（Full Request）：**基于Fork,发起一个请求PR,请求改动合并改进部分-接收请求--review--接受

**关注（Watch）：**关注后更新会有通知提醒

**事务卡片（Issue）：**发现代码Bug，但目前还没有成型代码，需要讨论问题issues时用

**Github主页**

**仓库主页**

**个人主页**

**注册Github账号**

github.com

翻墙Shadowsock

私有仓库--收费

**验证邮箱通过**

Resend重发邮件验证

QQ邮箱白名单--设置--反垃圾--设置域名白名单--inout里加入github.com

-----------基本概念，注册Github------------

# 创建仓库，仓库主页说明

-----------创建仓库，仓库主页说明--------

**创建仓库**

一个项目对应一个仓库

仓库名称--一般为项目名

用户名/仓库吗=名

**仓库描述**

readme说明文档--详细描述

**Github主页**

左边动态；右边关注收藏项目

**仓库主页**

Wacth关注

Star收藏

Fork复制克隆

edit--编辑项目描述

Create new file创建文件

Upload files上传文件

Find file搜索仓库

Clone or download直接下载或通过git克隆

**personal settings个人设置**

简介

账号设置

邮箱

通知中心

账单

SSH密钥

安全

应用

代码应用

组织

-----------创建仓库，仓库主页说明--------

# 仓库管理

-----------------仓库管理----------------

## 新建文件：

文件名需要加后缀；文件内容；描述文件；

编辑文件、删除文件：单击文件名--单击描述文件可查看文件详细信息--修改/删除

**上传文件:**Upload files--choose files

**搜索仓库文件:**Find file--仓库名/搜索信息(快捷键t-搜索文件)

**下载/检出项目:**Clone or download--下载项目代码

**Watch关注**--查看关注-更新有动态通知

**收藏（Star）**--查看收藏

**Frok复制克隆项目：**显示fored form 项目来源，独立存在；

**Pull request发起请求：**基于Fork,Fork项目--修改自己仓库的项目代码--发起一个请求PR,请求改动合并改进部分--接收请求--review--等待处理--回复Thanks

-----------------仓库管理----------------

# Github Issues

-----------------Github Issues-----------

发现代码Bug，但目前还没有成型代码，需要讨论问题issues时用

单击Issues--提交使用问题或建议或想法--显示所有的Bug或交流问题的列表

提交一个issues--接收--处理--回复

-----------------Github Issues-----------

# 基本概念实战操作

--------------基本概念实战操作-----------

以上总结:

--------------基本概念实战操作-----------

# 安装Git

-----------------安装Git-----------------

通过git管理github托管项目代码

安装git客户端

选择使用命令行环境

检验是否安装成功：

在桌面鼠标右键，出现Git GUI Here和Git Bash Here即说明安装成功

-----------------安装Git-----------------

# Git基本工作流程

----------------Git基本工作流程----------

Git 有三种状态，你的文件可能处于其中之一：

已修改（modified）、已暂存（staged）和已提交（committed）。

--已修改表示修改了文件，但还没跟踪，如新增的文件和刚修改过的文件。

--已暂存表示对一个已修改文件的当前版本做了标记（**git add**）。

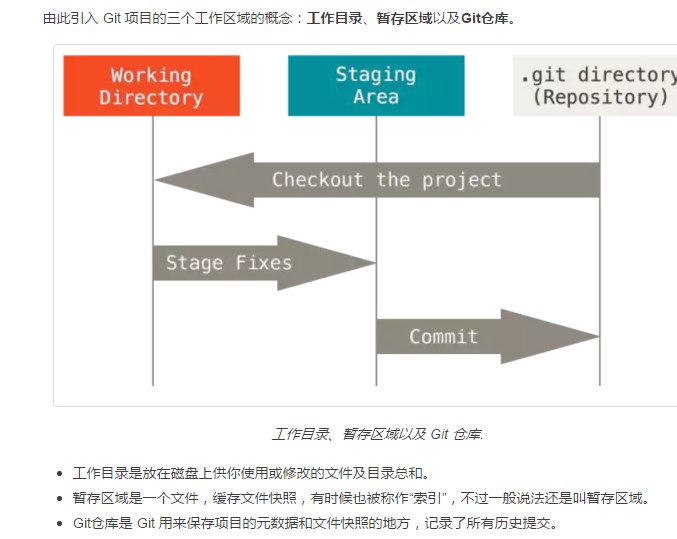
--已提交表示数据已经安全地保存在本地数据库中(**git commit**)。

由此引入 Git 项目的三个工作区域的概念：**工作目录、暂存区域以及Git仓库。untracked—Staged--commited**

**Git Repository(Git 仓库)：**最终确定的文件保存到仓库，比别人可见

**暂存区：**暂存已经修改的文件最统一提交到git仓库中

**Git工作区（Working Directory）：**添加、编辑、修改文件等动作



命令行--Linux

## 向仓库中添加文件流程：

--Git工作区(git add 文件名；git status)

--暂存区(git commit-m "提交描述"；git status）

-Git 仓库（git status）

----------------Git基本工作流程----------

# Git初始化及仓库创建和各种操作

---------Git初始化及仓库创建和操作-------

## 1.初始化用户名和邮箱

**设置用户名：**

git config --global user.name '用户名'

git config --global user.name 'hdabing'

(注意：git config --**global**表示本机所有的Git仓库都会使用这个配置， 只需要运行一次)

**设置用户名邮箱：**

git config --global user.email '邮箱地址'

git config --global user.email '13667831232@163.com'

（注意：该设置在Github上有动态提示）

**文本编辑器**

既然用户信息已经设置完毕，你可以配置默认文本编辑器了，当 Git 需要你输入信息时会调用它。 如果未配置，Git 会使用操作系统默认的文本编辑器，通常是 Vim。 如果你想使用不同的文本编辑器，例如 Emacs，可以这样做：git config --global core.editor emacs

## 2.查看设置

如果想要检查你的配置，可以使用 **git config --list** 命令来列出所有 Git 当时能找到的配置。你可以通过输入 git config <key>： 来检查 Git 的某一项配置,如：git config user.name

**检查当前文件状态：**

要查看哪些文件处于什么状态，可以用 git status 命令；status命令不仅列出了文件状态，还对下一步应该使用的命令给出了提示，很棒！

## 3.初始化一个新的Git仓库

ls/pwd/cd/

1.新建文件夹：create 文件夹名

2.在文件夹内初始化git(**创建git仓库**)：**git init**

3.向仓库中添加文件（3步）

（1）创建文件：touch 文件名；git status查看状态

（2）添加文件到暂存区：git add 文件名；git status 查看状态

（3）将文件从暂存区提交到仓库：git commit-m "提交描述"；git status查看状态

**# 在当前目录新建一个Git代码库**

**$git init**

**# 新建一个目录，将其初始化为Git代码库**

**git init [project-name]**

## 4.修改仓库文件（3步）

（1）编辑文件内：vi 文件名；查看文件内容：cat 文件名

（2）添加文件到暂存区：git add 文件名；git status 查看状态

（3）将文件从暂存区交到仓库：git status--git commit -m "提交描述"；git status查看状态

## 5.删除仓库文件（3步）

（1）删除文件：rm -f 文件名;h或rm 文件名

（2）从Git中上删除文件：git rm 文件名；git status 查看状态

（3）提交操作git commit -m "提交描述"；git status查看状态

## 6. 忽略某些文件

一般我们总会有些文件无需纳入 Git 的管理。 通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。

更典型的场景是，我们获得了一份JAVA工程源码，在本地我们需要用eclipse来进行开发，eclipse会为项目生成.classpath .project .settings等eclipse特有的文件（夹），还会自动编译源码，产出的.class文件放在bin目录下。这几个文件和bin文件夹，都没必要进行控制，因为他们会自动生成，不同的开发环境和编译环境的产出并不相同，我们只需把src目录下所有东西用git管理起来就行。

在这种情况下，我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。 来看一个实际的例子：

\*.class # 忽略clsss文件

# Mobile Tools for Java (J2ME)

.mtj.tmp/

# Package Files #

\*.jar #忽略jar包

\*.war

\*.ear

# eclipse files #

.project

.classpath

.settings/

GitHub 有一个十分详细的针对数十种项目及语言的 .gitignore 文件列表，你可以在 <https://github.com/github/gitignore> 找到它.

## 7. git push/pull 无需输入账户密码的设置方法

需要在本地创建SSH key，然后将生成的SSH key文件内容添加到github帐号上去。创建SSH key的方法很简单，执行如下命令就可以：

**ssh-keygen**

然后系统提示输入文件保存位置等信息，连续敲三次回车即可，生成的SSH key文件保存在中～/.ssh/id\_rsa.pub；然后用文本编辑工具打开该文件，我用的是vim,所以命令是：

**vim ~/.ssh/id\_rsa.pub**

接着拷贝.ssh/id\_rsa.pub文件内的所以内容，将它粘帖到github帐号管理中的添加SSH key界面中。

**打开github帐号管理中的添加SSH key界面的步骤如下：**

1. 登录github

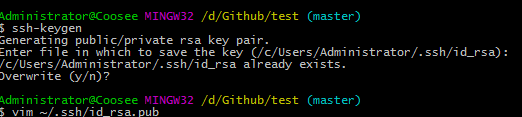
2. 点击右上方的Accounting settings图标

3. 选择 SSH key

4. 点击 Add SSH key

在出现的界面中填写SSH key的名称，填一个你自己喜欢的名称即可，然后将上面拷贝的~/.ssh/id\_rsa.pub文件内容粘帖到key一栏，在点击“add key”按钮就可以了。

例如：



复制文件内容：ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDYNwYPOTtAesGDI0q/UG1xu8OB1s9ib/wGzyX4UFTU6Db/12qWJYcvQuHtNpcfzzB4ExRgMbWw92JjO3kx9tiGunX/77vYMvoA2lCRT7etlJz6vRMeTFOUGXhOLH230ra6/Ebt73bG2buksuGJQbY0seIKVmBFDPlKyd6LPEr4OZU4Q4yKqe4/YlZ6+Z8xjbFL83XNP23zgaA2eyrzEfuBwZ5ObrBU40Y5HtgD1ub7fsLeTh1D10w6RsMBSTecouXeL74bVoivz/fUDAL8TvmAiUlHAWtMaBlI3HGoP+JiPGEzOXSTjNQzFzcDPFhJjsxqWmvFmROvTk9Rmw+tPLRX Administrator@Coosee

-------Git初始化及仓库创建和操作---------

## 8. 查看提交历史与回退版本：git log

**git log**命令显示从最近到最远的显示日志，git log 选项最常用的如下：

**-p**，用来显示每次提交的内容差异。 你也可以加上 -2 来仅显示最近两次提交如果你想看到每次提交的简略的统计信息，你可以使用 --stat 选项.

**--pretty**。这个选项可以指定使用不同于默认格式的方式展示提交历史。 这个选项有一些内建的子选项供你使用。比如用 oneline 将每个提交放在一行显示，查看的提交数很大时非常有用。另外还有short，full 和 fuller 可以用，展示的信息或多或少有些不同，请自己动手实践一下看看效果如何。

**git log**

**git log –pretty=oneline**

**回退版本：**

git reset –hard 版本号

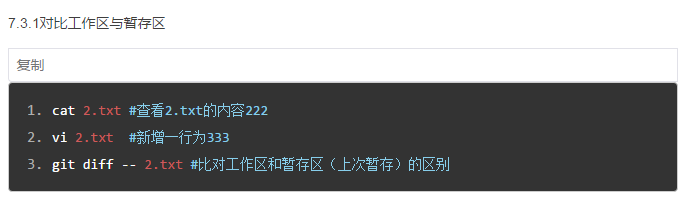
想使用版本回退操作可以使用如下2种命令，第一种是：**git reset –hard HEAD^** 那么如果要回退到上上个版本只需把HEAD^ 改成 **HEAD^^** 以此类推。那如果要回退到前100个版本的话，使用上面的方法肯定不方便，我们可以使用下面的简便命令操作：**git reset –hard HEAD~100** 即可。

**获取到版本号：git reflog**



## 9. 文件对比：

**（1）对比工作区与暂存区：git diff -filename**



**（2）对比暂存区和仓库区：git diff --cached -- filename**



**（3）对比对比工作区和仓库区：git diff HEAD~ filename**



## 10. Git撤销修改和删除文件操作

**（1）撤销修改**

有如下几种方法可以做修改：

**第一：**如果我知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后add添加到暂存区，最后commit掉。

**第二：**我可以按以前的方法直接恢复到上一个版本。使用 **git reset –hard HEAD^**

**第三：git checkout - file** 可以丢弃工作区的修改；在工作区做的修改全部撤销。注意：命令git checkout - file 中的 - 很重要，如果没有 - 的话，那么命令变成创建分支了。

**（2）删除文件操作**

1）git rm file:删除文件并同时提交到仓库

2）rm file;git commit –m :第一步删除文件但仓库记录没有删除，还需要执行commit命令

## 11. 创建与合并分支

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

重命名本地分支：

git branch -m <old\_branch\_name> <new\_branch\_name>

重命名远程分支：

在本地的clone版本中重命名分支

删除远程待修改的分支名

则本地的新分支名push到远程

## 12. 分支管理策略

1)创建一个dev分支：

git branch dev

git checkout dev

git checkout –b dev

2)修改readme.txt内容：

3)添加到暂存区：

4)切换回主分支(master)：git checkout master

5)合并dev分支，使用命令：

git merge dev或git merge –no-ff -m “注释”dev:

6)查看历史记录：git log –graph --pretty=online --abbrev-commit

7）删除分支：git branch –d dev

**分支策略：**

首先master主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的dev分支上干活，干完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

**解决分支冲突？**

合并分支同一处相同时会冲突；提示Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，其中<<<HEAD是指主分支修改的内容，>>>>>fenzhi1 是指fenzhi1上修改的内容，修改查看cat file；git add file;git commit –m ‘提交描述’；即可。

**bug分支:**

在开发中，会经常碰到bug问题，那么有了bug就需要修复，在Git中，分支是很强大的，每个bug都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时的分支删除掉。如果g修复需要很长时间或工作时间问题无法及时提交时，可以先隐藏起来：**git stash**；等以后恢复现场后继续工作。

**例子：**

git branch bug-404

git checkout bug-404

或git checkout –b bug-404

…bug修复…完成…add;commit后

git checkout master

git merge –no-ff –m “合并描述”bug-404

git branch –d bug-404

git stash

查看隐藏工作区：**git stash list**

Git把stash内容存在某个地方了，但是需要恢复一下，可以使用如下2个方法：

(1)**git stash apply**恢复，恢复后，stash内容并不删除，你需要使用命令g**it stash drop**来删除。

(2)另一种方式是使用**git stash pop**,恢复的同时把stash内容也删除了。

## 13. 多人协作

方式当你从远程库克隆时候，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且远程库的默认名称是origin。

多人协作工作模式一般是这样的：

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

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

3)如果合并有冲突，则需要解决冲突，并在本地提交。再用git push origin branch-name推送。

1）查看远程库的信息：**git remote**

2）查看远程库的详细信息：**git remote –v**

**一、推送分支：**

推送分支就是把该分支上所有本地提交到远程库中，推送时，要指定本地分支，这样，Git就会把该分支推送到远程库对应的远程分支上：

**git push origin master**

**git push origin dev**

注意：master分支是主分支，因此要时刻与远程同步。一些修复bug分支不需要推送到远程去，可以先合并到主分支上，然后把主分支master推送到远程去。

**二、抓取分支：**

多人协作时，大家都会往master分支上推送各自的修改。

1. 首先要克隆远程仓库到本地；

git clone https://github.com/hdabing/git.git

1. 小伙伴要在某个分支（如dev分支）上做开发，就必须把远程的origin的dev分支到本地来，于是可以使用命令**创建远程origin的dev分支到本地来：git checkout –b dev origin/dev**
2. …**开发**…完成…把现在的dev分支推送到远程去：**git push origin dev**
3. **出现冲突:**修改同一个地方--先指定本地dev分支与远程origin/dev分支的链接后更新最新仓库：**git branch --set-upstream dev origin/dev;git pull**—手动解决冲突，提交—**git push origin dev**

## 14. Git 的使用场景

**场景1：暂存文件以及取消已暂存的文件**

git add file

git commit –m “提交描述”

git rm –cached file

**场景2：修改已提交文件以及取消已暂存的内容**

修改内容—add—commit—push

git reset Head file

HEAD是一个特别的指针，指向你正在工作的本地分支。

**场景3：修改文件以及撤销修改内容**

git checkout – file(一旦撤销，无法恢复)

**概念区分：fetch vs pull**

**fetch**命令只是将远端数据拉到本地仓库，并不自动合并到当前工作分支。若要合并，还需手动合并。例如，执行git fetch origin，就会抓取自上次克隆以来别人上传到此远程仓库中的所有更新。

**pull**命令则除了会抓取数据，还能将远端分支自动合并到本地仓库中当前分支。

**场景4：撤销提交（revert/reset）**

git revert <commit>

git revert HEAD~n

如果是revert当前提交，则不需要HEAD后的~n。

git reset <commit>

git reset HEAD~n

与revert不同的是，需要指定为：HEAD~1

revert命令可以撤销已经提交的快照，但它并不会将该提交从项目的提交历史中移除，而是会判断要撤销的这次提交引入了哪些变化，然后将此变化撤销（此次撤销事实上还是一种变化），再将这次撤销作为一个提交。因此，在执行revert命令后，如果通过git log查看提交历史，可以看到会新增一个revert提交。

默认情况下，如果使用reset命令，会将当前的分支回退到指定commit，然后自指定commit到最新commit之间的内容会放在工作目录下，使得我们可以再提交。

**注意：**

针对远程的提交记录，应尽量避免使用git reset命令。倘若在本地进行了reset之后，又进行了另外的修改并提交。此时，本地的提交记录与远程的提交记录在reset的那个点产生了分叉。此时，如果执行git push，会在本地合并后提交，并同步远程提交记录。则团队其他成员会因为这个变化的提交记录而困惑。由于一部分变更消失，甚至可能导致一些数据被破坏。因此，使用reset命令要格外当心，通常情况，应尽量针对本地提交（未push到远程）进行reset。优先考虑使用revert命令。

## 15. Git处理发布

**（1） 添加标签**

**发布的时候，添加标签，便于查找：git tag git\_1.0 dev**

以上命令中的两个参数分别指明了标签的名称（1.0）和希望打标签的点（RB\_1.0分支的末梢（所对应的版本或者说所对应的提交）

**查看版本库在的标签列表：git tag**

**（2） 变基（**与合并分支相似**）：**把一条分支上的修改在另一条分支的末梢重现；

git checkout master

git rebase dev

**（3）打补丁**

如果没有了发布分支，如何给1.0.x分支打补丁呢？很简单，只需要在打标签的地方再创建一条分支即可。前面创建分支的时候，命令的最后一个参数是新分支的父分支名称，现在只须把父分支名称改成发布标签名即可。命令如下：

git branch git\_1.1 git\_1.0

git checkout git\_1.1

**（4）为代码发布创建归档文件**

没有必要总是把历史记录（也就是Git版本库）一起发布，通常情况下，将标签对应的版本内容打包成一个tar包或者zip包就足够了。

　　Git提供了**git archive**命令来做归档处理。

**创建tar文件：**

**git archive --format=tar --prefix=mysite-1.0/ git\_1.0 |gzip > mysite-1.0.tar.gz**

　　该命令中有三个参数：

　　--format指明要产生tar格式的输出。

　　--prefix指明包中所有东西都放到mysite-1.0/目录下。

　　git\_1.0指明要归档的标签的名称。

　　最后一段命令把git archive产生的tar文件用管道输出的方法传递给命令gzip进行压缩，而压缩结果则重定向到mysite-1.0.tar.gz压缩包里。

**创建zip文件：**

**git archive --format=zip –prefix=mysite-1.0/ git\_1.0 >mysite-1.0.zip**

生成zip格式和tar格式的命令参数几乎一样，只是改变了传递给--format的参数，而且无需通过命令gzip管道输出，直接把归档内容保存到归档文件中。

**（5）**

# Git远程管理仓库

--------------Git远程管理仓库----------------

作用：备份；实现代码共享集中化管理

## 1.将远程git克隆到本地仓库操作

（将远程的项目代码下载到本地）：git clone 仓库地址url

**# 下载一个项目和它的整个代码历史，创建一个本地仓库的克隆版本：**

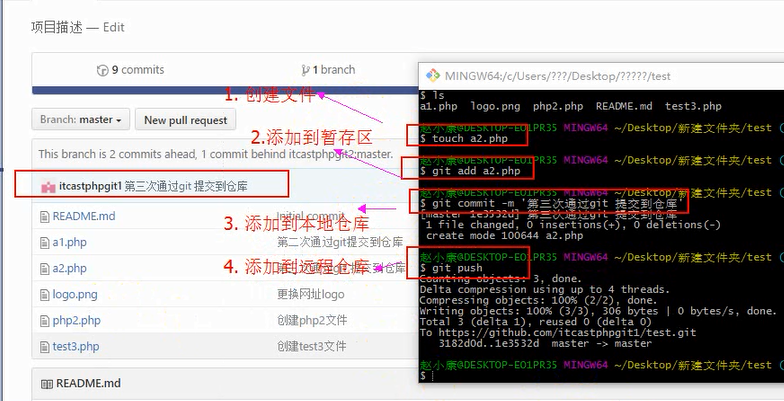
git clone [url]

git clone /path/to/repository

例如：git clone <https://github.com/hdabing/test.git>

**# 如果是远端服务器上的仓库：**

git clone username@host:/path/to/repository



## 2.将本地仓库同步到远程仓库：git push

在本地的git仓库下运行命令：**git remote add origin** [**https://github.com/hdabing/git.git**](https://github.com/hdabing/git.git)

修改、删除等本地操作后同步到远程仓库：**git push –u origin master**

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。

需要编辑配置文件：vi .git/config

需要输入账号密码，才能同步

合并现在的提交到上一次提交：git commit –amend

**注意：**可以添加ssh-key方式不用每次push都要输入账号、密码；如果远程仓库有文件未更新，需有执行**git push origin master**更新远程仓库内容到本地仓库（

<http://www.crifan.com/git_github_git_push_origin_master_error_failed_to_push_some_refs_to/>）

无法同步问题



--------------Git远程管理仓库----------------

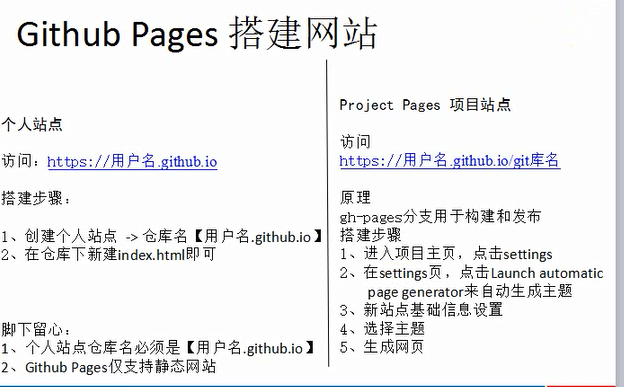
# Github Pages 搭建网站

-----------Github Pages 搭建网站--------

新建仓库：仓库名/用户名.github.io

新建文件：index.html

访问：<https://hdabing.github.io>



-----------Github Pages 搭建网站--------

# Git 基本常用命令

mkdir： XX (创建一个空目录 XX指目录名)

pwd： 显示当前目录的路径。

git init 把当前的目录变成可以管理的git仓库，生成隐藏.git文件。

git add XX 把xx文件添加到暂存区去。

git commit –m “XX” 提交文件 –m 后面的是注释。

git status 查看仓库状态

git diff XX 查看XX文件修改了那些内容

git log 查看历史记录

git reset –hard HEAD^ 或者 git reset –hard HEAD~ 回退到上一个版本 (如果想回退到100个版本，使用git reset –hard HEAD~100 )

cat XX 查看XX文件内容

git reflog 查看历史记录的版本号id

git checkout — XX 把XX文件在工作区的修改全部撤销。

git rm XX 删除XX文件

git remote add origin https://github.com/tugenhua0707/testgit

关联一个远程库

git push –u(第一次要用-u 以后不需要) origin master

把当前master分支推送到远程库

git clone https://github.com/tugenhua0707/testgit 从远程库中克隆

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

git branch 查看当前所有的分支

git checkout master 切换回master分支

git merge dev 在当前的分支上合并dev分支

git branch –d dev 删除dev分支

git branch name 创建分支

git stash 把当前的工作隐藏起来 等以后恢复现场后继续工作

git stash list 查看所有被隐藏的文件列表

git stash apply 恢复被隐藏的文件，但是内容不删除

git stash drop 删除文件

git stash pop 恢复文件的同时 也删除文件

git remote 查看远程库的信息

git remote –v 查看远程库的详细信息

git push origin master Git会把master分支推送到远程库对应的远程分支上