# 命令 git clone url 克隆一个本地库

$ git clone git@github.com:michaelliao/gitskills.git

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (3/3), done.

$ cd gitskills

$ ls

README.md

# 生成一个新的SSH。

$ssh-keygen -t rsa -C "email"

之后直接回车，不用填写东西。之后会让你输入密码。然后就生成一个目录.ssh ，里面有两个文件：id\_rsa , id\_rsa.pub

把这个SSH放到github上。用公钥。先在GitHub上注册一个用户，然后进入account-setting ，把id\_rsa.pub的内容复制进去就可以了。

# 安装 [Git](http://lib.csdn.net/base/28)

sudo apt-get install git

查看git配置  
git config --list  
或者～/gedit .gitconfig

git设置代理（一般用于公司内部网络，如果上网正常，不需要设置）  
git config --global http.proxy ip:port  
git用户名  
git config --global user.name "FIRST\_NAME LAST\_NAME"  
配置email  
git config --global user.email "MY\_NAME@example.com"

建立Bitbucket用户和仓库  
https://confluence.atlassian.com/display/BITBUCKET/Create+a+Bitbucket+account+and+a+Git+repository

初始  
mkdir /path/to/your/project  
cd /path/to/your/project  
git init  
git remote add origin https://XXX@bitbucket.org/YYYY/ZZZZ.git  
git add .  
git commit -m "modify"  
git push -u origin master

修改后  
修改文件（git已经建立）  
git add .  
git commit -m "注释"  
git push -u origin master

检出  
git pull

克隆  
git clone https://XXX@bitbucket.org/YYYY/ZZZZ.git

**GitHub 教程系列文章**：

GitHub 使用教程图文详解  <http://www.linuxidc.com/Linux/2014-09/106230.htm>

Git 标签管理详解 <http://www.linuxidc.com/Linux/2014-09/106231.htm>

Git 分支管理详解 <http://www.linuxidc.com/Linux/2014-09/106232.htm>

Git 远程仓库详解 <http://www.linuxidc.com/Linux/2014-09/106233.htm>

Git 本地仓库（Repository）详解 <http://www.linuxidc.com/Linux/2014-09/106234.htm>

Git 服务器搭建与客户端安装  <http://www.linuxidc.com/Linux/2014-05/101830.htm>

Git 概述 <http://www.linuxidc.com/Linux/2014-05/101829.htm>

# 提交并指定路径

git push origin master:master

# 拉代码 git pill

# git status

命令可以列出当前目录所有还没有被git管理的文件和被git管理且被修改但还未提交(git commit)的文件

# 添加修改文件 git add .

# 查看历史提交 git log

# Commit提交

git commit -m '测试'

# 推送远程仓库

git push

# 推送远程仓库并指定目录

git push origin master:master

# 强制提交

git push -f命令去强制提交

# 退回操作

git reset --hard f093b6ed512f761a346e2e5c0f00230e448c217c

(1) Git checkout

恢复某个已修改的文件（撤销未提交的修改）：

$ git checkout file-name

例如：git checkout src/com/Android/.../xxx.Java

比如修改的都是java文件，不必一个个撤销，可以使用

$ git checkout \*.java

撤销所有修改

$ git checkout .

(2) git revert

撤销某次操作，此次操作之前和之后的commit和history都会保留，并且把这次撤销

作为一次最新的提交

\* git revert HEAD 撤销前一次 commit

\* git revert HEAD^ 撤销前前一次 commit

\* git revert commit-id （比如：fa042ce57ebbe5bb9c8db709f719cec2c58ee7ff）撤销指定的版本，撤销也会作为一次提交进行保存。

git revert是提交一个新的版本，将需要revert的版本的内容再反向修改回去，版本会递增，不影响之前提交的内容

git help <command> # 显示command的help

git show # 显示某次提交的内容 git show $id

git co -- <file> # 抛弃工作区修改

git co . # 抛弃工作区修改

git add <file> # 将工作文件修改提交到本地暂存区

git add . # 将所有修改过的工作文件提交暂存区

git rm <file> # 从版本库中删除文件

git rm <file> --cached # 从版本库中删除文件，但不删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset -- . # 从暂存区恢复到工作文件

git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git ci <file> git ci . git ci -a # 将git add, git rm和git ci等操作都合并在一起做　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　git ci -am "some comments"

git ci --amend # 修改最后一次提交记录

git revert <$id> # 恢复某次提交的状态，恢复动作本身也创建次提交对象

git revert HEAD # 恢复最后一次提交的状态

查看文件diff

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <id1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

查看提交记录

git log git log <file> # 查看该文件每次提交记录

git log -p <file> # 查看每次详细修改内容的diff

git log -p -2 # 查看最近两次详细修改内容的diff

git log --stat #查看提交统计信息

tig

Mac上可以使用tig代替diff和log，brew install tig

Git 本地分支管理

查看、切换、创建和删除分支

git br -r # 查看远程分支

git br <new\_branch> # 创建新的分支

git br -v # 查看各个分支最后提交信息

git br --merged # 查看已经被合并到当前分支的分支

git br --no-merged # 查看尚未被合并到当前分支的分支

git co <branch> # 切换到某个分支

git co -b <new\_branch> # 创建新的分支，并且切换过去

git co -b <new\_branch> <branch> # 基于branch创建新的new\_branch

git co $id # 把某次历史提交记录checkout出来，但无分支信息，切换到其他分支会自动删除

git co $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支

git br -d <branch> # 删除某个分支

git br -D <branch> # 强制删除某个分支 (未被合并的分支被删除的时候需要强制)

分支合并和rebase

git merge <branch> # 将branch分支合并到当前分支

git merge origin/master --no-ff # 不要Fast-Foward合并，这样可以生成merge提交

git rebase master <branch> # 将master rebase到branch，相当于： git co <branch> && git rebase master && git co master && git merge <branch>

Git补丁管理(方便在多台机器上开发同步时用)

git diff > ../sync.patch # 生成补丁

git apply ../sync.patch # 打补丁

git apply --check ../sync.patch #测试补丁能否成功

Git暂存管理

git stash # 暂存

git stash list # 列所有stash

git stash apply # 恢复暂存的内容

git stash drop # 删除暂存区

Git远程分支管理

git pull # 抓取远程仓库所有分支更新并合并到本地

git pull --no-ff # 抓取远程仓库所有分支更新并合并到本地，不要快进合并

git fetch origin # 抓取远程仓库更新

git merge origin/master # 将远程主分支合并到本地当前分支

git co --track origin/branch # 跟踪某个远程分支创建相应的本地分支

git co -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上

git push # push所有分支

git push origin master # 将本地主分支推到远程主分支

git push -u origin master # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

Git远程仓库管理

git remote -v # 查看远程服务器地址和仓库名称

git remote show origin # 查看远程服务器仓库状态

git remote add origin git@ github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址(用于修改远程仓库地址) git remote rm <repository> # 删除远程仓库

创建远程仓库

git clone --bare robbin\_site robbin\_site.git # 用带版本的项目创建纯版本仓库

scp -r my\_project.git git@ git.csdn.net:~ # 将纯仓库上传到服务器上

mkdir robbin\_site.git && cd robbin\_site.git && git --bare init # 在服务器创建纯仓库

git remote add origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址

git push -u origin master # 客户端首次提交

git push -u origin develop # 首次将本地develop分支提交到远程develop分支，并且track

git remote set-head origin master # 设置远程仓库的HEAD指向master分支

也可以命令设置跟踪远程库和本地库

git branch --set-upstream master origin/master

git branch --set-upstream develop origin/develop

# [Git 常用命令](http://www.cnblogs.com/gabo/p/3902644.html)

git init  　　　　　　　　　创建repository

git add readme.txt 　　　git add命令实际上就是把要提交的所有修改放到暂存区（Stage）

add所有文件：git add .

add并rm所有：git add -A

提交modify和rm（新加的文件不会自动提交）：git commit -a

git commit -m "log"　　　提交到master，添加log内容

git status 查看repository 状态

git diff readme.txt　　　　查看修改内容

git log (--prety=oneline)　查看log，后面参数可选

git reset --hard HEAD^　 回退上个版本（HEAD指向master的指针，^个数表示上几个版本）

git reset --hard HEAD~100 回退n个版本

git reset --hard 3628164　　回退到commit为3628164版本

git reflog　　　　　　　　　查看每一次命令

git checkout --readme.txt 把readme.txt文件在工作区的修改全部撤销

这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

git checkout -- file命令中的“--”很重要，没有“--”，就变成了“创建一个新分支”的命令

git reset HEAD readme.txt 可以把暂存区的修改撤销掉（unstage），重新放回工作区

git rm readme.txt　　　　从版本库中删除文件

git checkout -- readme.txt 如果在文件管理器中误删了文件，因为版本库里还有呢，可以用这句恢复

关联一个远程库，使用命令git remote add origin https://github.com/HelloGabo/MyUnityTest.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

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

克隆repository。创建一个空的repository，然后用命令

git clone https://github.com/HelloGabo/gitskills.git

将现在的repository克隆进去

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（Fast forward模式，这种模式下删除分支后，会丢失分支信息）

 -no-ff参数，表示禁用“Fast forward”  会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

git merge --no-ff -m "merge with no-ff" dev

删除分支：git branch -d name

git branch -D branch-name 强行删除分支（当在分支上修改后，不想并入其他分支，想删掉，就要强行删除）

解决分支合并产生的冲突：

手动修改冲突后提交就行

git log --graph 命令可以看到分支合并图

eg：git log --graph --pretty=oneline --abbrev-commit

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue -101来修复它，但是，等等，当前正在dev上进行的工作还没有提交

stash功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：git stash（有修改还没add并commit的时候调用后，现场保留，分支清空）

git stash list命令查看工作现场

恢复有两个办法：

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

查看远程库的信息，用git remote

git remote -v显示更详细的信息

$ git remote -v  
origin https://github.com/HelloGabo/MyUnityTest.git (fetch)　　抓取  
origin https://github.com/HelloGabo/MyUnityTest.git (push)　　推送

显示了可以抓取和推送的origin的地址。如果没有推送权限，就看不到push的地址

推送分支

推送分支，就是把该分支上的所有本地提交推送到远程库。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

如果要推送其他分支，比如dev，就改成：

git push orgin dev

抓取分支

小伙伴要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是他用这个命令创建本地dev分支：

git checkout -b dev origin/dev

现在，就可以在dev上继续修改，然后，时不时地把dev分支push到远程：

git push origin dev

碰巧你也对同样的文件作了修改，并试图推送：推送失败，因为你的小伙伴的最新提交和你试图推送的提交有冲突

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

git pull

失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

git branch --set-upstream dev origin/dev

再pull就行了，但合并会有冲突。需要手动解决

标签管理

发布一个版本时，我们通常先在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。

Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针（跟分支很像对不对？但是分支可以移动，标签不能移动），所以，创建和删除标签都是瞬间完成的。

创建标签

切换到需要打标签的分支上git checkout dev

1、敲命令git tag name就可以打一个新标签：git tag v1.0

2、git tag查看所有标签

默认标签是打在最新提交的commit上的。有时候，如果忘了打标签，比如，现在已经是周五了，但应该在周一打的标签没有打，怎么办？

方法是找到历史提交的commit id，然后打上就可以了：

git log --pretty=oneline --abbrev-commit

对应的commit id是“ef51d63”，敲入命令git tag v0.9 ef51d63

3、可以用git show tagname查看标签信息：git show v0.9

4、可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

git tag -a v0.1 -m "version 0.1 released" 02172e4

5、可以通过-s用私钥签名一个标签：git tag -s v0.2 -m "signed version 0.2 released" a8093be

签名采用PGP签名，因此，必须首先安装gpg（GnuPG），如果没有找到gpg，或者没有gpg密钥对，就会报错

用命令git show tagname可以看到PGP签名信息

用PGP签名的标签是不可伪造的，因为可以验证PGP签名。验证签名的方法比较复杂

操作标签

标签打错了，也可以删除：git tag -d v1.0

推送某个标签到远程，使用命令git push origin tagname：

一次性推送全部尚未推送到远程的本地标签：git push origin --tags

从远程删除。删除命令也是git push origin :refs/tags/tagname

忽略特殊文件

有些时候，你必须把某些文件放到Git工作目录中，但又不能提交它们，比如保存了数据库密码的配置文件啦，等等，每次git status都会显示“Untracked files ...”

这个问题解决起来也很简单，在Git工作区的根目录下创建一个特殊的.gitignore文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件

不需要从头写.gitignore文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

最后一步就是把.gitignore也提交到Git

比如，忽略Python编译产生的.pyc、.pyo、dist等文件或目录：

# Python:

\*.py[cod]

\*.so

\*.egg

\*.egg-info

dist

build

使用Windows的童鞋注意了，如果你在资源管理器里新建一个.gitignore文件，它会非常弱智地提示你必须输入文件名，但是在文本编辑器里“保存”或者“另存为”就可以把文件保存为.gitignore了。

Git 中的文件忽略  
1. 共享式忽略新建 .gitignore 文件，放在工程目录任意位置即可。.gitignore 文件可以忽略自己。忽略的文件，只针对未跟踪文件有效，对已加入版本库的文件无效。  
2. 独享式忽略针对具体版本库 ：.git/info/exclude针对本地全局： git config --global core.excludefile ~/.gitignore  
忽略的语法规则：  
(#)表示注释  
(\*) 表示任意多个字符;   
(?) 代表一个字符;  
 ([abc]) 代表可选字符范围  
如果名称最前面是路径分隔符 (/) ，表示忽略的该文件在此目录下。  
如果名称的最后面是 (/) ，表示忽略整个目录，但同名文件不忽略。  
通过在名称前面加 (!) ，代表不忽略。  
例子如下：  
# 这行是注释  
\*.a # 忽略所有 .a 伟扩展名的文件  
!lib.a # 但是 lib.a 不忽略，即时之前设置了忽略所有的 .a  
/TODO # 只忽略此目录下 TODO 文件，子目录的 TODO 不忽略   
build/ # 忽略所有的 build/ 目录下文件  
doc/\*.txt # 忽略如 doc/notes.txt, 但是不忽略如 doc/server/arch.txt

配置别名

如果敲git st就表示git status那就简单多了

1、只需要敲一行命令，告诉Git，以后st就表示status：git config --global alias.st status

-2、-global参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

3、命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区。既然是一个unstage操作，就可以配置一个unstage别名：

git config --global alias.unstage 'reset HEAD'

敲入命令：git unstage test.py

实际上Git执行的是：git reset HEAD test.py

4、配置一个git last，让其显示最后一次提交信息：git config --global alias.last 'log -l'

用git last就能显示最近一次的提交

甚至还有人丧心病狂地把lg配置成了：

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"

搭建git服务器

搭建Git服务器需要准备一台运行Linux的机器，强烈推荐用Ubuntu或Debian，这样，通过几条简单的apt命令就可以完成安装。

假设你已经有sudo权限的用户账号，下面，正式开始安装。

第一步，安装git：sudo apt-get install git

第二步，创建一个git用户，用来运行git服务：sudo adduser git

第三步，创建证书登录：

收集所有需要登录的用户的公钥，就是他们自己的id\_rsa.pub文件，把所有公钥导入到/home/git/.ssh/authorized\_keys文件里，一行一个。

第四步，初始化Git仓库：

先选定一个目录作为Git仓库，假定是/srv/sample.git，在/srv目录下输入命令：sudo git init --bare sample.git

Git就会创建一个裸仓库，裸仓库没有工作区，因为服务器上的Git仓库纯粹是为了共享，所以不让用户直接登录到服务器上去改工作区，并且服务器上的Git仓库通常都以.git结尾。然后，把owner改为git：

sudo chown -R git:git sample.git

第五步，禁用shell登录：

出于安全考虑，第二步创建的git用户不允许登录shell，这可以通过编辑/etc/passwd文件完成。找到类似下面的一行：

git:x:1001:1001,,,:/home/git:/bin/bash

改为：

git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell

这样，git用户可以正常通过ssh使用git，但无法登录shell，因为我们为git用户指定的git-shell每次一登录就自动退出。

第六步，克隆远程仓库：

现在，可以通过git clone命令克隆远程仓库了，在各自的电脑上运行：

git clone git@server:/srv/sample.git

剩下的推送就简单了

管理公钥

如果团队很小，把每个人的公钥收集起来放到服务器的/home/git/.ssh/authorized\_keys文件里就是可行的。如果团队有几百号人，就没法这么玩了，这时，可以用[Gitosis](https://github.com/res0nat0r/gitosis)来管理公钥。

这里我们不介绍怎么玩[Gitosis](https://github.com/res0nat0r/gitosis)了，几百号人的团队基本都在500强了，相信找个高水平的Linux管理员问题不大。

管理权限

有很多不但视源代码如生命，而且视员工为窃贼的公司，会在版本控制系统里设置一套完善的权限控制，每个人是否有读写权限会精确到每个分支甚至每个目录下。因为Git是为Linux源代码托管而开发的，所以Git也继承了开源社区的精神，不支持权限控制。不过，因为Git支持钩子（hook），所以，可以在服务器端编写一系列脚本来控制提交等操作，达到权限控制的目的。[Gitolite](https://github.com/sitaramc/gitolite)就是这个工具。

国外网友制作的Git Cheat Sheet，建议打印出来备用：

[Git Cheat Sheet](http://www.git-tower.com/blog/assets/2013-05-22-git-cheat-sheet/cheat-sheet-large01.png)

Git的官方网站：[http://git-scm.com](http://git-scm.com/)，英文自我感觉不错的童鞋，可以经常去官网看看

# gerrit标签管理(tag)

发布一个版本时，我们通常先在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的时刻的历史版本取出来。所以，标签也是版本库的一个快照。

Git的标签虽然是版本库的快照，但其实它就是指向某个commit的指针（跟分支很像对不对？但是分支可以移动，标签不能移动），所以，创建和删除标签都是瞬间完成的。

首先在gerrit 对应项目上添加以下权限：

|  |  |
| --- | --- |
| **Reference:** | refs/tags/\* |

*Create Reference*

*Forge Committer Identity*

*Push*

*Push Annotated Tag*

*例如：*



创建标签

切换到需要打标签的分支上git checkout branches

1、敲命令git tag name就可以打一个新标签：git tag v1.0

2、git tag查看所有标签

默认标签是打在最新提交的commit上的。有时候，如果忘了打标签，比如，现在已经是周五了，但应该在周一打的标签没有打，怎么办？

方法是找到历史提交的commit id，然后打上就可以了：

git log --pretty=oneline --abbrev-commit

对应的commit id是“ef51d63”，敲入命令git tag v0.9 ef51d63

3、可以用git show tagname查看标签信息：git show v0.9

4、可以创建带有说明的标签，用-a指定标签名，-m指定说明文字：

git tag -a v0.1 -m "version 0.1 released" 02172e4

5、可以通过-s用私钥签名一个标签：git tag -s v0.2 -m "signed version 0.2 released" a8093be

签名采用PGP签名，因此，必须首先安装gpg（GnuPG），如果没有找到gpg，或者没有gpg密钥对，就会报错

用命令git show tagname可以看到PGP签名信息

用PGP签名的标签是不可伪造的，因为可以验证PGP签名。验证签名的方法比较复杂

操作标签

标签打错了，也可以删除：git tag -d v1.0

推送某个标签到远程，使用命令git push origin tagname：

一次性推送全部尚未推送到远程的本地标签：git push origin --tags

从远程删除。删除命令也是git push origin :refs/tags/tagname

#### Push Annotated Tag

允许推送带注释的标签到远程仓库, 标签必须带有注释.

#创建带注释的标签

git tag -a <tagname> -m <comment>

#提交标签到远程仓库

git push origin --tags

#获取标签

git fetch --tags

标签的email一定会与提交者的邮箱进行验证，如果推送其他人的标签需要同时赋予Push Annotated Tag和Forge Committer Identity权限.

git show v1.0

tag v1.0

Tagger: username <email>3

如果需要推送轻量级标签(不带注释), 给refs/tags/\*命名空间赋予Create Reference权限, 轻量级标签就像Git中的分支一样.   
如果需要删除或者重写标签, 给refs/tags/\*命名空间赋予带force选项的Push权限,删除标签需要和删除分支一样的权限.

#### Push Signed Tag

允许推送PGP签证的标签到远程仓库

git tag -u "gpg-key-id" -m "tag comment" <tagname>