**Git 操作手册**

注意：编辑工具用notepad++ UTF-8 without BOM 模式，老外东西切记目录不能含有中文。

备注：

cd：切换目录

mkdir：创建目录

pwd：查看当前目录

**安装**

linux安装git

1. Linux（新的Debian或Ubuntu Linux）
   1. Sudo apt-get install git
2. linux（旧的Debian或Ubuntu Linux）
   1. sudo apt-get install git-core
3. linux其他版本
   1. 先从Git官网下载源码，然后解压，依次输入：./config，make，sudo make install这几个命令安装。

Mac os x安装git

1. 一是安装homebrew，然后通过homebrew安装Git，具体方法请参考homebrew的文档：<http://brew.sh/>。
2. 第二种方法更简单，也是推荐的方法，就是直接从AppStore安装Xcode，Xcode集成了Git，不过默认没有安装，你需要运行Xcode，选择菜单“Xcode”->“Preferences”，在弹出窗口中找到“Downloads”，选择“Command Line Tools”，点“Install”就可以完成安装了。

Windows 安装git

msygit 安装

msysgit是Windows版的Git，从http://msysgit.github.io/下载，然后按默认选项安装即可。

Git 官网安装

<http://www.git-scm.com/>

**初始化**

$ git config --global user.name "Your Name"

$ git config --global user.email [email@example.com](mailto:email@example.com)

因为Git是分布式版本控制系统，所以，每个机器都必须自报家门：你的名字和Email地址。

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置

**创建版本库**

1. 新建空目录test

mkdir test

cd test

pwd

1. 变成git管理的仓库

Git init

注：细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

Ls –ah：查看当前目录所有文件，包含隐藏的



**文件保存到库**

1. git add (file)文件添加到库

git add test.txt

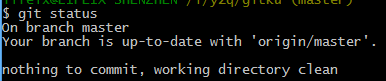
1. git commit 添加的文件全部提交到库里

git commit –m “memo:content”

-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

**查看库的状态**

Git status

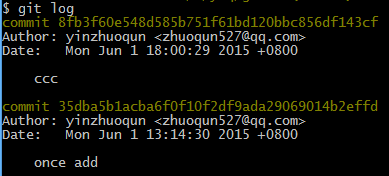


**查看文件修改的内容**

Git diff (filename)

**查看历史文件修改的历史记录**

Git log



git log --pretty=oneline（关键历史信息查看）



注意：按 q 退出end

**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

**回退版本**

git reset --hard HEAD^

**滚回最新版本**

git reset --hard 3628164

命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是3628164...

git reflog用来记录你的每一次命令

**工作区和暂存区**



**丢弃工作区的修改git checkout – file**

$ git checkout -- readme.txt

让这个文件回到最近一次git commit或git add时的状态。

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 file可以把暂存区的修改撤销掉（unstage），重新放回工作区

**删除文件**

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：

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

**远程仓库**

**添加远程仓库**

**克隆远程仓库**

Git clone <https://github.com/yinzhuoqun/yzq>

**查看远程库的信息**

$ git remote

Origin

$ git remote -v

origin git@github.com:michaelliao/learngit.git (fetch)

origin git@github.com:michaelliao/learngit.git (push)

**推送分支到远程库**

（master分支）$ git push origin master

（dev分支）$ git push origin dev

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

**从远程抓取分支**

Git pull （branchname）

**创建分支**

创建dev分支，然后切换到dev分支：

$ git checkout -b dev

Switched to a new branch 'dev

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

**查看分支**

Git brach

**合并分支**

git merge命令用于合并指定分支到当前分支。

$ git merge dev

Updating d17efd8..fec145a

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：

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

Merge made by the 'recursive' strategy.

readme.txt | 1 +

1 file changed, 1 insertion(+)

**删除分支**

合并完成后，就可以放心地删除dev分支了：

$ git branch -d dev

Deleted branch dev (was fec145a).

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。



带参数的git log也可以看到分支的合并情况：

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

\* 59bc1cb conflict fixed

|\

| \* 75a857c AND simple

\* | 400b400 & simple

|/

\* fec145a branch test

...

**分支策略**



**Stash储藏**

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

Saved working directory and index state WIP on dev: 6224937 add merge

HEAD is now at 6224937 add merge

用git stash list命令看看stash

用git stash apply恢复

用git stash drop来删除

用git stash pop，恢复的同时把stash内容也删

**标签管理-创建标签（默认是最新的commit）**

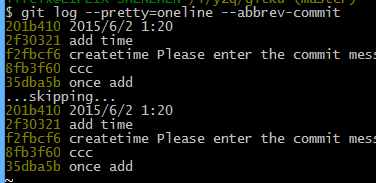
命令git tag <name>就可以打一个新标签

$ git tag v1.0

历史commit打标签

git tag查看标签信息

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



比方说要对add merge这次提交打标签，它对应的commit id是6224937，敲入命令：

$ git tag v0.9 6224937

再用命令git tag查看标签

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" 3628164

git show <tagname>查看标签说明信息

私钥签名（暂不讨论）

标签管理-删除标签（-d）

$ git tag -d v0.1

Deleted tag 'v0.1' (was e078af9)

推送标签至远程

git push origin <tagname>

$ git push origin v1.0

Total 0 (delta 0), reused 0 (delta 0)

To git@github.com:michaelliao/learngit.git

\* [new tag] v1.0 -> v1.0

推送全部标签至远程

$ git push origin --tags

Counting objects: 1, done.

Writing objects: 100% (1/1), 554 bytes, done.

Total 1 (delta 0), reused 0 (delta 0)

To git@github.com:michaelliao/learngit.git

\* [new tag] v0.2 -> v0.2

\* [new tag] v0.9 -> v0.9

删除远程标签

先从本地删除再从远程删除

$ git tag -d v0.9

Deleted tag 'v0.9' (was 6224937)

$ git push origin :refs/tags/v0.9

To git@github.com:michaelliao/learngit.git

- [deleted] v0.9

**自定义git**

让Git显示颜色，会让命令输出看起来更醒目：

$ git config --global color.ui true

**忽略特殊文件**

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。

# Windows:

Thumbs.db

ehthumbs.db

Desktop.ini

# Python:

\*.py[cod]

\*.so

\*.egg

\*.egg-info

dist

build

# My configurations:

db.ini

deploy\_key\_rsa

配置别名

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

$ git config --global **alias**.co checkout

$ git config --global **alias**.ci commit

$ git config --global **alias**.br branch

--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。

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'

配置一个git last，让其显示最后一次提交信息：

$ git config --global **alias**.last 'log -1'

这样，用git last就能显示最近一次的提交：

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：

$ su**do** apt-**get** install git

第二步，创建一个git用户，用来运行git服务：

$ su**do** 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：

$ su**do** 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：

$ su**do** 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.gitCloning into 'sample'...warning: You appear to have cloned an empty repository.

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" \t "http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)来管理公钥。

这里我们不介绍怎么玩[Gitosis](https://github.com/res0nat0r/gitosis" \t "http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)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" \t "http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)就是这个工具。

这里我们也不介绍[Gitolite](https://github.com/sitaramc/gitolite" \t "http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/_blank)了，不要把有限的生命浪费到权限斗争中。

Git 环境生成公钥命令:

ssh-keygen

git@ubuntu:~$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/git/.ssh/id\_rsa):

Created directory '/home/git/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again: