# Git 使用简介

工欲善其事，必先利其器。本文简要涉及Git基础、安装、配置以及日常使用过程中的常见操作，所有操作均基于Git命令行，当然也可以在流行的编辑器如Eclipse、Idea、Xcode中已图形化的方式使用Git。

符号约定：说明本文中使用的符号对应的意义

● <>：小于号和大于号表示这是一个变量，使用时应该替换为具体内容

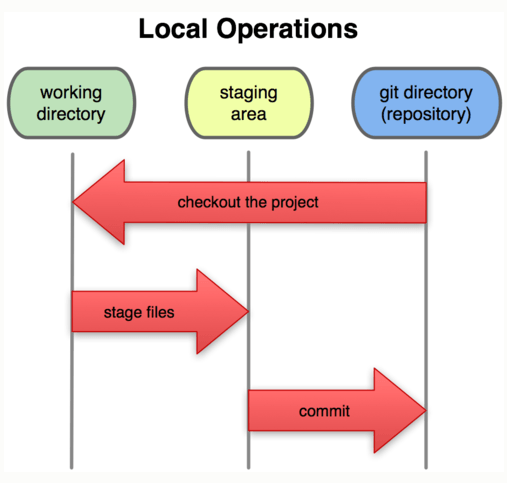
● //：注释语句

## Git基础

### 文件的三种状态

在 Git 内都只有三种状态：**已提交**（committed），**已修改**（modified）和**已暂存**（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。

由此可以看到 Git 管理项目时，文件流转的三个工作区域：Git 的工作目录，暂存区域，以及本地仓库。



基本的 Git 工作流程如下：

1、在工作目录中修改某些文件。

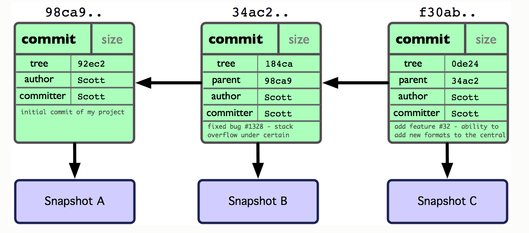
2、对修改后的文件进行快照，然后保存到暂存区域。

3、提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。

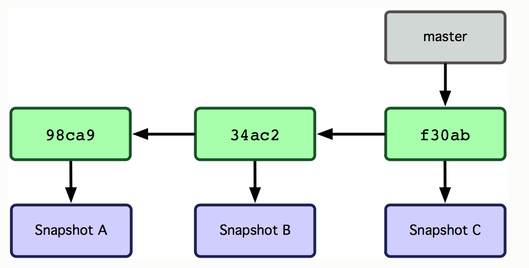
所以，可以从文件所处的位置来判断状态：如果是 Git 目录中保存着的特定版本文件，就属于已提交状态；如果作了修改并已放入暂存区域，就属于已暂存状态；如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

### 分支

在 Git 中提交时，会保存一个提交（commit）对象，该对象包含一个指向暂存内容快照的指针，包含本次提交的作者等相关附属信息，包含零个或多个指向该提交对象的父对象指针：首次提交是没有直接祖先的，普通提交有一个祖先，由两个或多个分支合并产生的提交则有多个祖先。



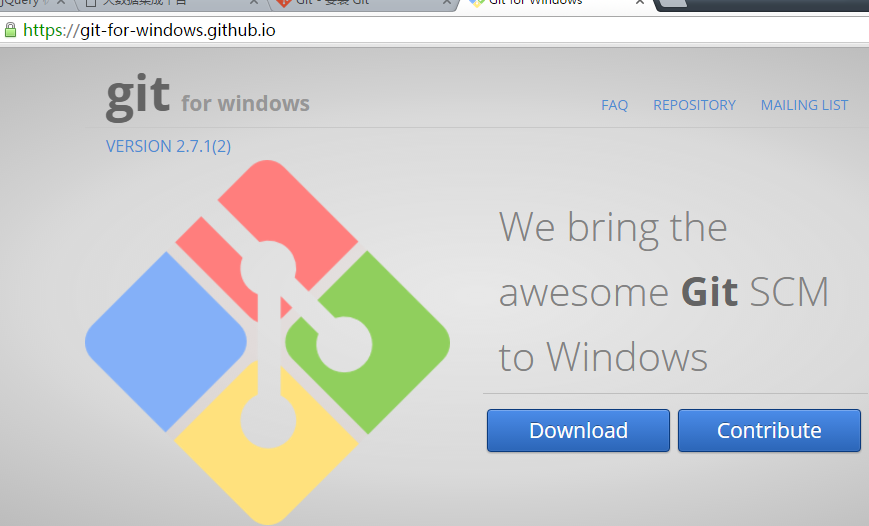
Git 中的分支，其实本质上仅仅是个指向 commit 对象的可变指针。Git 会使用 master 作为分支的默认名字。在若干次提交后，你其实已经有了一个指向最后一次提交对象的 master 分支，它在每次提交的时候都会自动向前移动。



## 安装Git

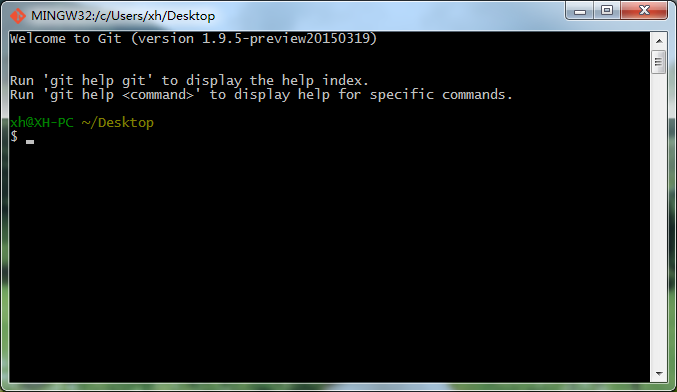
### windows

到<https://git-for-windows.github.io/>下载最新版



下载完毕双击安装即可。

安装后，可在需要使用Git的地方右击鼠标，选择Git Bash Here（或Git Bash）即可调出Git命令行。



PS：其他平台请自行Google

## 配置

### 用户信息

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

$ git config --global user.name "John Doe"

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

PS:

1. 如果用了 --global 选项，那么更改的配置文件就是位于你用户主目录下的那个，以后你所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。
2. 要检查已有的配置信息，可以使用 git config --list 命令：

### 生成 SSH 公钥

大多数 Git 服务器都会选择使用 SSH 公钥来进行授权。系统中的每个用户都必须提供一个公钥用于授权，没有的话就要生成一个。生成公钥的过程在所有操作系统上都差不多。 首先先确认一下是否已经有一个公钥了。SSH 公钥默认储存在账户的主目录下的 ~/.ssh 目录。进去看看：

$ cd ~/.ssh

$ ls

authorized\_keys2 id\_dsa known\_hosts

config id\_dsa.pub

关键是看有没有用 something 和 something.pub 来命名的一对文件，这个 something 通常就是id\_dsa 或 id\_rsa。有 .pub 后缀的文件就是公钥，另一个文件则是密钥。假如没有这些文件，或者干脆连 .ssh 目录都没有，可以用 ssh-keygen 来创建。该程序在 Linux/Mac 系统上由 SSH 包提供，而在 Windows 上则包含在 MSysGit 包里：

使用ssh-keygen可以生成sshkey。

$ ssh-keygen

Generating public/private rsa key pair.

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

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /Users/schacon/.ssh/id\_rsa.

Your public key has been saved in /Users/schacon/.ssh/id\_rsa.pub.

The key fingerprint is:

43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a schacon@agadorlaptop.local

它先要求你确认保存公钥的位置（.ssh/id\_rsa），然后它会让你重复一个密码两次，如果不想在使用公钥的时候输入密码，可以留空。

现在，所有做过这一步的用户都得把它们的公钥给你或者 Git 服务器的管理员（假设 SSH 服务被设定为使用公钥机制）。他们只需要复制 .pub 文件的内容然后发邮件给管理员。公钥的样子大致如下：

$ cat ~/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU

GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3

Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA

t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En

mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx

NrRFi9wrf+M7Q== schacon@agadorlaptop.local

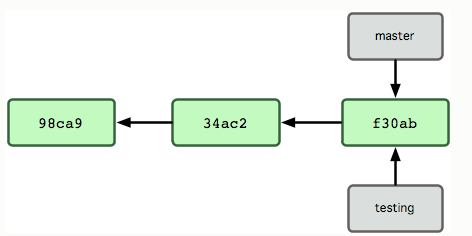
## 常用操作

### 分支

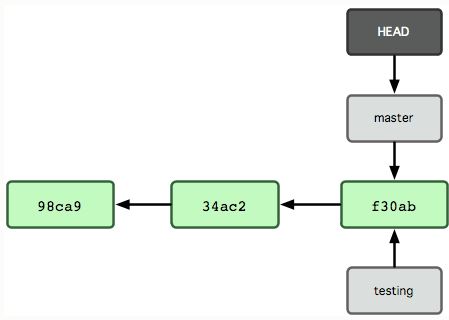
#### 新建分支

语法：git branch <branchname>

如 新建testing分支：git branch testing这会在当前commit对象上新建要给分支



当存在多个分支时Git通过保存一个名为HEAD的特别指针，来标示你当前工作在哪一个分支。



运行 git branch 命令，仅仅是建立了一个新的分支，但不会自动切换到这个分支中去，所以当前依然还在 master 分支里工作。

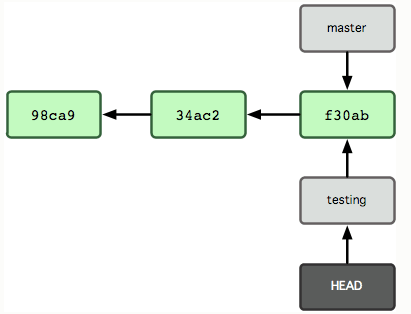
#### 切换分支

语法: git checkout <branchname>

如切换到testing分支

$ git checkout testing

这样HEAD就指向了testing分支



随着每一次的提交，HEAD也会一直向前移动。

PS：可以通过git checkout –b <branchname>新建并切换到该分支

#### 删除分支

语法: git branch –d <branchname>

如删除testing分支

$ git branch –d testing

#### 合并分支

在开发过程中，你可能正工作在dev分支上，这时boss找到你，让你去修复一个bug。此时你就可以新建一个分支去修复这个bug，假设你新建bug01分支。在这个分支上，你很快修复了这个bug，并熟练使用git add、git commit 命令提交该代码，此时你需要将这个补丁添加到dev分支上。此时可以使用git merge 命令。此时你需要先切换到dev分支:git checkout dev，将bug01分支上的代码合并到dev分支上:git merge bug01。

### 解决冲突

有时候，合并分支并不是那么容易的，在一个团队中，如果其他人和你正在修改同一个文件的内容，后提交的那个人将出现冲突。此时Git将产生冲突的文件列出来，等待你手动的解决冲突。以下是Git显示冲突文件的样式

<<<<<<< HEAD

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>> testing

可以看到=======隔开的上半部分，是HEAD中的内容，下半部分是testing上的内容。

解决冲突的办法无非是二者选其一或者由你亲自整合到一起。比如你可以通过把这段内容替换为下面这样来解决：

<div id="footer">

please contact us at email.support@github.com

</div>

这个解决方案各采纳了两个分支中的一部分内容，而且我还删除了 <<<<<<<，======= 和 >>>>>>> 这些行。

解决冲突后，你需要重新运行git add 和 git commit 命令将这些修改提交到本地Git仓库。

PS：

分支的其他操作

git branch -r 列出远程分支，例如：

#git branch -r

m/master -> origin\_apps/m1\_2.3.4

origin\_apps/hardware/test

origin\_apps/m1

origin\_apps/m1\_2.3.4

origin\_apps/master

git branch -a 列出本地分支和远程分支，例如：

#git branch -a

\* master

newbranch

remotes/m/master -> origin\_apps/m1\_2.3.4

remotes/origin\_apps/hardware/test

remotes/origin\_apps/m1

remotes/origin\_apps/m1\_2.3.4

remotes/origin\_apps/master

git branch -d | -D <branchname> //删除branchname分支，例如

git branch –d testing

### 拉取远程代码到本地

语法: git fetch <remote name> <branch name>

拉取后，远程仓库中的代码存在<remote name>/<branchname>上，如拉取远程Git仓库origin上的dev分支上的代码:

$ git fetch origin dev

拉取代码后常见的操作就是合并远程的代码到对应的本地分支上，此时使用

$ git merge origin/dev

**PS：**

git允许设置本地分支追踪(track)远程分支，使用

$ git branch --set-upstream <local branch name> <remote name>/<branch name>

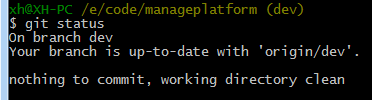
如设置本地master分支追踪远程origin仓库中的dev分支

$ git branch --set-upstream master origin/dev

设置追踪关系后，可以使用git pull命令达到git fetch加git merge的效果，即使用git pull后，Git会拉取（fetch）当前所在分支追踪的远程分支的代码，并自动merge到当前所在分支。当然，若此时发生冲突，你还是需要手动的去解决冲突。关于如何解决冲突，上文已经提及，此处不再赘述。

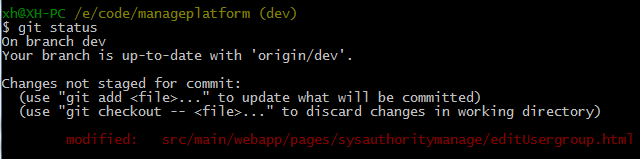
### 推送本地代码到远程服务器

在拉取远程Git仓库的代码后，此时所有的文件都是已提交(committed)的状态，此时使用git status命令将显示如下



表示我们的工作目录时clean的。

当我们修改了某个文件并保存(Ctrl+S)后，，此时使用git status，将显示你修改了哪些文件



可以看到此时的文件处于已修改（modified）状态，若想提交这次修改，你需要使用git add <file>命令将这些文件变为已暂存（staged）状态，再使用git commit命令将这些已暂存状态的文件变为已提交的状态。注意，commit之后，这些文件只是提交到了本地Git仓库，并没有推送（push）到远程Git仓库。若想推送到远程仓库，你需要使用

git push <remote name> <local branch name>:<remote branch name>命令，但通常在一个团队中会有多个人在同时进行开发，所以在你push之前，你通常应该先拉取(fetch)远程Git上最新的代码，并合并(merge)到本地对应分支后再推送(push)到远程Git仓库。如果你忘记拉取了服务器上最新的代码并合并到本地，而直接推送(push)，git会拒绝(reject)你的推送，并提示你当前代码落后于Git仓库的版本(**当然，如果你觉得远程的Git仓库代码有问题或有错误，此时你不想merge服务器上最新的代码而直接提交本地代码，你可以使用git命令强制本地代码覆盖服务器代码**，后面会讲述)。根据上文的叙述，通常在开发过程中，提交代码的步骤通常分为以下几步

git status //查看哪些文件被修改

git add <file> //将修改的文件变为已暂存状态，可以使用git add //<directoy>/\* 提交某目录下的所有修改的文件，或者使用git //add –A，提交所有修改

git commit –m “comments” //将已暂存状态的文件变为已提交状态，双引号中填//写本次提交说明

git pull //拉取远程代码并合并到本地，前提是设置了//track关系

git push //将本地代码推送到远程Git仓库，前提是设置了//track关系

### 新建本地项目并推送到远程Git仓库

以上讲的都是在已使用Git管理的项目中操作，那若果有一个没有使用Git管理的项目，想要使用Git管理，该怎么做？，其实很简单。

准备条件:

1. 新建一个文件夹作为项目的根目录，如demo
2. 在远程服务器上新建一个仓库，并命名，假设新建好的仓库的地址是https://github.com/johnxue2013/ocdemo.git
3. 进入步骤一中新建的文件夹demo，添加需要添加的文件，如README.md，右击选择git bash here（windows上是这样操作，如果在mac或者linux上，需要从终端进入）

执行如下命令:

$ git init

$ git status

$ git add README.md

$ git commit -m "first commit"

$ git remote add origin https://github.com/johnxue2013/ocdemo.git

$ git push -u origin master

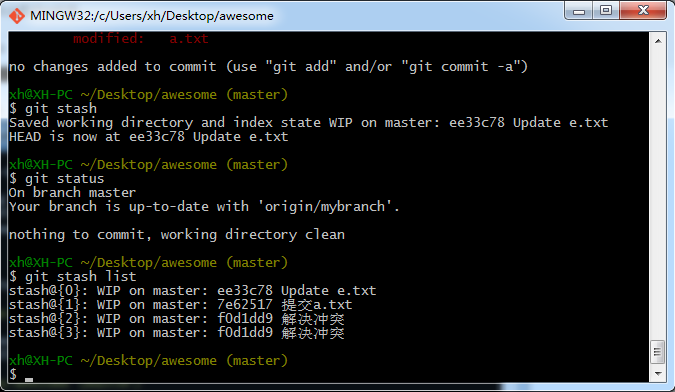
执行后，本地Git仓库的数据，将被推送到远程Git仓库。以上命令中git init会初始化一个git项目，并新建.git文件，该文件存储仓库的版本等信息。git status命令将显示所有未add或commit 的文件，git add README.md 将README.md 将文件保存到暂存区，git commit -m "first commit" 将文件保存到本地Git仓库，git remote add origin <https://github.com/johnxue2013/ocdemo.git> 添加远程Git仓库的地址，并将远程仓库命名为origin，git push -u origin master 将本地Git的内容推送到远程Git仓库。此处将推送到远程服务器的master分支。-u参数意为指定origin远程仓库为默认远程仓库（一个本地Git仓库可以同时和多个远程Git仓库保持同步）

### 储藏(git stash)

假设一个场景：当你在项目的dev分支上正在开发一个新功能时，突然接到boss说在holiday分支上有一个bug需要立即修复，但此时你工作在dev分支上，并且你正在开发的功能还没写完，无法使用，不想提交(commit)代码，此时你又需要切换到holiday分支上，你熟练的使用git checkout holiday命令切换到holiday分支，此时你会发现，你切换不过去，Git提示当前分支有未commit的内容，无法切换分支。此时就可以使用git stash。

“储藏”可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

git stash 之后你就可以切换到其他分支去工作了，当你在其他分支的工作完成时，可以切换到之前使用git stash的分支，并使用git stash apply命令，恢复到你离开之前的样子。你也可以多次使用git stash，每一次stash都会存储起来。你可以使用git stash list命令查看所有stash记录。



你也可以使用git stash apply stash@{<number>}，去指定恢复哪一次stash。默认情况下git stash apply 使用最近一次stash进行恢复。你也可以使用git stash save “coments” 为stash添加注释。

PS：git stash apply只是尝试应用存储的工作，stash的内容仍然在栈上，要移除它使用git stash drop, 你也可以使用git stash pop来重新应用存储，同时将其从栈中移除。移除所有stash 使用git stash clear。

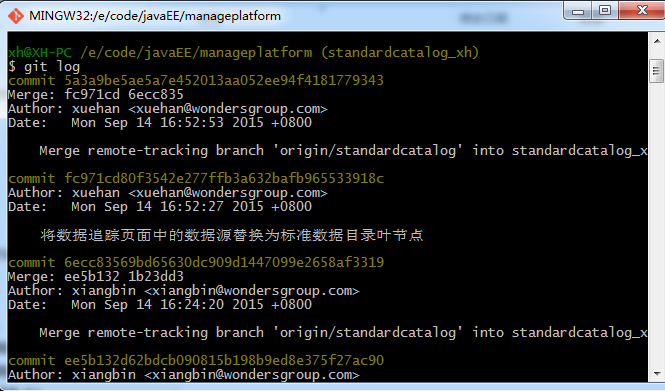
### 强制本地代码覆盖远程Git代码

该操作比较暴力，应谨慎使用

 git push <remote name> <local branch name>:<remote branch name> --force

### 版本回退

每一次的提交都会产生一个commit ID，只要找到你想回退的版本的commit ID，就可以查看当时的版本。你可以使用git log，查看每一次提交。



图中黄色部分就是commit ID，获取commit ID后，使用

git reset --hard <commit\_id>

进行回退，此时的回退只是本地仓库的回退，若要同时回退服务器版本使用

git push <remote name> HEAD –force

如 git push origin HEAD –force

### 忽略文件

在日常开发中，有一些文件可能不需要追踪该文件内容，此时可以忽略该文件。

如果想忽略某些文件或文件夹且该文件或文件夹从未被提交过，可以直接在项目根目录下在git命令行窗口中使用touch .gitignore命令新建一个文件，在该文件中添加想要忽略的文件或文件夹， 如忽略项目根目录下的.idea文件夹，则.gitignore文件的内容应该是下面这个样子。

/.idea

PS：有时候不小心提交了不想提交的文件，如某些配置文件。当提交之后，再在.gitignore配置忽略将不起作用，因为该文件只能用作与Untrack Files， 也就是从来没有被git记录过的文件(自添加以后，从未add以及commit过的文件)。也就是说一旦add或者commit之后，该文件或者文件夹就处于track file， 此时再修改.gitignore当然就不起作用了。此时正确的做法是使用 git rm --cached <文件名> 然后修改.gitignore配置忽略，最后 git add .gitignore 再 git commit –m "注释" 再 git push <服务器别名> <本地分支名>:<服务器分支名> 至此服务器将不包含被忽略的文件或者文件夹。

说明: git rm --cached <文件名> 删除的是追踪状态，而不是物理文件；如果你真不想要了，你可以使用直接使用rm 命令删除该文件或文件夹

### git使用https方式连接，在命令行模式下记住密码

使用https方式连接进行命令行操作，如git fetch、git push时，经常需要输入用户名和密码，这是一个很不方便的地方，此时可以通过设置让Git记住用户名和密码。

在命令行中输入以下命名并回车

git config credential.helper store

设置后，只要再推送一次，以后就不需要用户名和密码了 只要运行后，下次push的时候再输入一次密码，git就会记住。

PS：如果你想要深入Git，请阅读Pro Git。



以上就是全部内容，enjoy Git

2016-2-23 11:54:35