

从0开始学习 GitHub 系列之「Git速成」

原创 2016-05-29 stormzhang AndroidDeveloper



阅读本文大概需要 6 分钟。

前面的 GitHub 系列文章介绍过，GitHub 是基于 Git 的，所以也就意味着 Git 是基础，如果你不会 Git，那么接下来你完全继续不下去，所以今天的教程就来说说 Git，当然关于 Git 的知识单凭一篇文章肯定说不完的，我这篇文章先介绍一些最基本的、最常用的一些 Git 知识，争取让你们 Git 速成。

1/ 什么是Git？

Git 是 Linux 发明者 Linus 开发的一款新时代的版本控制系统，那什么是版本控制系统呢？怎么理解？网上一大堆详细的介绍，但是大多枯燥乏味，对于新手也很难理解，这里我只举几个

例子来帮助你们理解。

熟悉编程的知道，我们在软件开发中源代码其实是最重要的，那么对源代码的管理变得异常重要：

比如为了防止代码的丢失，肯定本地机器与远程服务器都要存放一份，而且还需要有一套机制让本地可以跟远程同步；

又比如我们经常是好几个人做同一个项目，都要对一份代码做更改，这个时候需要大家互不影响，又需要各自可以同步别人的代码；

又比如我们开发的时候免不了有bug，有时候刚发布的功能就出现了严重的bug，这个时候需要紧急对代码进行还原；

又比如随着我们版本迭代的功能越来越多，但是我们需要清楚的知道历史每一个版本的代码更改记录，甚至知道每个人历史提交代码的情况；

等等等类似以上的情况，这些都是版本控制系统能解决的问题。所以说，版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统，对于软件开发领域来说版本控制是最重要的一环，而 Git 毫无疑问是当下最流行、最好用的版本控制系统。

2/ Git 安装

上面说了，Git 是一个版本控制系统，你也可以理解成是一个工具，跟 Java 类似，使用之前必须先得下载安装，所以第一步必须要安装，我用的是 Mac，Mac 上其实系统自带 Git 的，不过这里统一提供一下各平台的安装方式，这部分就不过多介绍，相信大家这里搞的定。

Mac：<https://sourceforge.net/projects/git-osx-installer/>

Windows：<https://git-for-windows.github.io/>

Linux：`apt-get install git`

3/ 如何学习 Git？

安装好 Git 之后，怎么学习是个问题，其实关于 Git 有很多图形化的软件可以操作，但是我强烈建议大家从命令行开始学习理解，我知道没接触过命令行的人可能会很抵触，但是我的亲身实践证明，只有一开始学习命令行，之后你对 Git 的每一步操作才能理解其意义，而等你熟练之后你想用任何的图形化的软件去操作完全没问题。

我一开始教我们团队成员全是基于命令行的，事后证明他们现在已经深深爱上命令行无法自拔，他们很理解 Git 每一步操作的具体含义，以致于在实际项目很少犯错，所以我这里也是基于命令行去教你们学习理解。

4 Git 命令列表

怎么判断你 Git 有没有安装成功？请在命令行里输入 **git**，如果出现以下提示证明你已经安装成功了。



```
1. storm@192: ~/my_projects/stormzhang (zsh)
→ stormzhang git:(master) x git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

The most commonly used git commands are:
add          Add file contents to the index
bisect       Find by binary search the change that introduced a bug
branch       List, create, or delete branches
checkout     Checkout a branch or paths to the working tree
clone        Clone a repository into a new directory
commit       Record changes to the repository
diff         Show changes between commits, commit and working tree, etc
fetch        Download objects and refs from another repository
grep         Print lines matching a pattern
init         Create an empty Git repository or reinitialize an existing one
log          Show commit logs
merge        Join two or more development histories together
mv           Move or rename a file, a directory, or a symlink
pull         Fetch from and integrate with another repository or a local branch
push         Update remote refs along with associated objects
rebase       Forward-port local commits to the updated upstream head
reset        Reset current HEAD to the specified state
rm           Remove files from the working tree and from the index
show         Show various types of objects
status       Show the working tree status
tag          Create, list, delete or verify a tag object signed with GPG
```

Git 所有的操作命令开头都要以 **git** 开头，上面列举了最常用的一些 Git 命令，紧接着会有一句英文解释这个命令的意义，都不是很难的单词，不妨试着看一下，不过没有实际操作你仍然

不好理解，下面我们来以一个实际的操作来介绍下一些常用命令的含义。

5/ Git 具体命令

第一步，我们先新建一个文件夹，在文件夹里新建一个文件（我是用 Linux 命令去新建的，Windows用户可以自己手动新建）

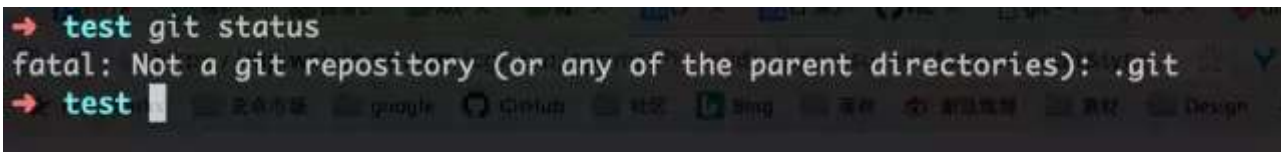
mkdir test （创建文件夹test）

cd test （切换到test目录）

touch a.md （新建a.md文件）

这里提醒下：在进行任何 Git 操作之前，都要先切换到 Git 仓库目录，也就是要先切换到项目的文件夹目录下。

这个时候我们先随便操作一个命令，比如 **git status**，可以看到如下提示（别纠结颜色之类的，配置与主题不一样而已）：

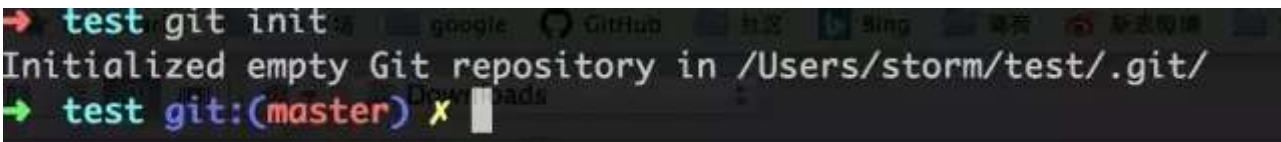


```
→ test git status
fatal: Not a git repository (or any of the parent directories): .git
→ test
```

意思就是当前目录还不是一个 Git 仓库。

git init

这个时候用到了第一个命令，代表初始化 git 仓库，输入 **git init** 之后会提示：



```
→ test git init
Initialized empty Git repository in /Users/storm/test/.git/
→ test git:(master)
```

可以看到初始化成了，至此 test 目录已经是一个 git 仓库了。

git status

紧接着我们输入 **git status** 命令，会有如下提示：


```

→ test git:(master) X git status
On branch master
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  a.md

nothing added to commit but untracked files present (use "git add" to track)

```

意思就是当前目录还不是一个 Git 仓库。

默认就直接在 master 分支，关于分支的概念后面会提，这时最主要的是提示 a.md 文件 Untracked files，就是说 a.md 这个文件还没有被跟踪，还没有提交在 git 仓库里呢，而且提示你可以使用 **git add <file>** 去操作你想要提交的文件。

git status 这个命令顾名思义就是查看状态，这个命令可以算是使用最频繁的一个命令了，建议大家没事就输入下这个命令，来查看你当前 git 仓库的一些状态。

git add

上面提示 a.md 文件还没有提交到 git 仓库里，这个时候我们可以随便编辑下 a.md 文件，然后输入 **git add a.md**，然后再输入 **git status**：

```

→ test git:(master) X git add a.md
→ test git:(master) X git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file:   a.md

```

git status 这个命令顾名思义就是查看状态，这个命令了，建议大家没事就输入下这个命令，来查看你当前 git 仓库的一些状态。

git add

上面提示 a.md 文件还没有提交到 git 仓库里，这个时候我们可以随便编辑下 a.md 文件，然后输入 git add a.md，然后再输入 git status：

此时提示以下文件 Changes to be committed，意思就是 a.md 文件等待被提交，当然你可以使用 **git rm --cached** 这个命令去移除这个缓存。

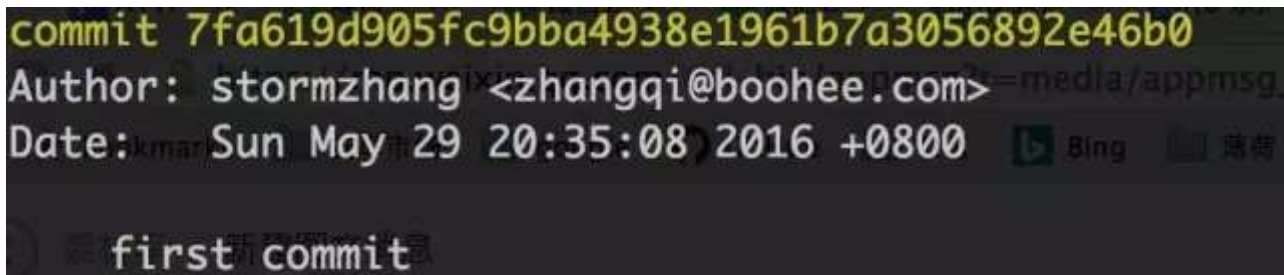
git commit

接着我们输入 **git commit -m 'first commit'**，这个命令什么意思呢？**commit** 是提交的意思，**-m** 代表是提交信息，执行了以上命令代表我们已经正式进行了第一次提交。

这个时候再输入 **git status**，会提示 nothing to commit。

git log

这个时候我们输入 **git log** 命令，会看到如下：



```
commit 7fa619d905fc9bba4938e1961b7a3056892e46b0
Author: stormzhang <zhangqi@boohee.com>
Date: Sun May 29 20:35:08 2016 +0800
    first commit
```

git log 命令可以查看所有产生的 commit 记录，所以可以看到已经产生了一条 commit 记录，而提交时候的附带信息叫 'first commit'。

git add & git commit

看到这里估计很多人会有疑问，我想要提交直接进行 commit 不就行了么，为什么先要再 add 一次呢？首先 git add 是先把改动添加到一个「暂存区」，你可以理解成是一个缓存区域，临时保存你的改动，而 git commit 才是最后真正的提交。这样做的好处就是防止误提交，当然也有办法把这两步合并成一步，不过后面再介绍，建议新手先按部就班的一步步来。

git branch


branch 即分支的意思，分支的概念很重要，尤其是团队协作的时候，假设两个人都在做同一个项目，这个时候分支就是保证两人能协同合作的最大利器了。举个例子，A, B两人都在做同一个项目，但是不同的模块，这个时候A新建了一个分支叫a，B新建了一个分支叫b，这样A、B做的所有代码改动都各自在各自的分支，互不影响，等到俩人都把各自的模块都做完了，最后再统一把分支合并起来。

执行 **git init** 初始化git仓库之后会默认生成一个主分支 master，也是你所在的默认分支，也基本是实际开发正式环境下的分支，一般情况下 master 分支不会轻易直接在上面操作的，你们可以输入 **git branch** 查看下当前分支情况：



```
→ test git:(master) git branch
* master
```

如果我们想在此基础上新建一个分支呢，很简单，执行 **git branch a** 就新建了一个名字叫 a 的分支，这时候分支 a 跟分支 master 是一模一样的内容，我们再输入 **git branch** 查看的当前分支情况：

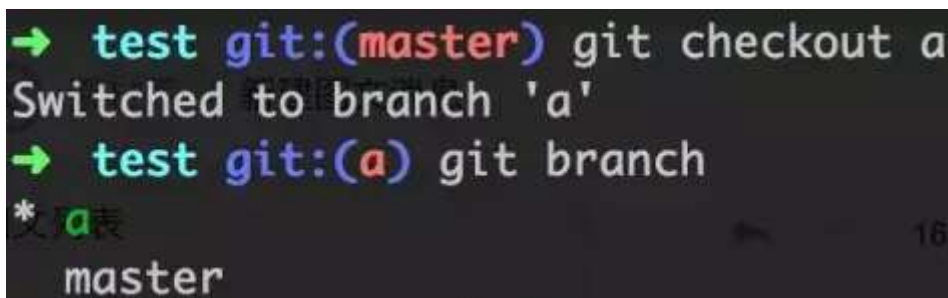


```
→ test git:(master) git branch
* master
```

但是可以看到 master 分支前有个 * 号，即虽然新建了一个 a 的分支，但是当前所在的分支还是在 master 上，如果我们想在 a 分支上进行开发，首先要先切换到 a 分支上才行，所以下一步要切换分支

git checkout a

执行这个命令，然后再输入 **git branch** 查看下分支情况：



```
→ test git:(master) git checkout a
Switched to branch 'a'
→ test git:(a) git branch
* a
master
```

可以看到当前我们在的分支已经是a了，这个时候 A 同学就可以尽情的在他新建的a分支去进行代码改动了。

那有人就说了，我要先新建再切换，未免有点麻烦，有没有一步到位的，聪明：

git checkout -b a

这个命令的意思就是新建一个a分支，并且自动切换到a分支。

git merge

A同学在a分支代码写的不亦乐乎，终于他的功能完工了，并且测试也都ok了，准备要上线了，这个时候就需要把他的代码合并到主分支master上来，然后发布。**git merge** 就是合并分支用到的命令，针对这个情况，需要先做两步，第一步是切换到 master 分支，如果你已经在了就不用切换了，第二步执行 **git merge a**，意思就是把a分支的代码合并过来，不出意外，这个时候a分支的代码就顺利合并到 master 分支来了。为什么说不出意外呢？因为这个时候可能会有冲突而合并失败，留个包袱，这个到后面进阶的时候再讲。

git branch -d

有新建分支，那肯定有删除分支，假如这个分支新建错了，或者a分支的代码已经顺利合并到 master 分支来了，那么a分支没用了，需要删除，这个时候执行 **git branch -d a** 就可以把a分支删除了。

git branch -D

有些时候可能会删除失败，比如如果a分支的代码还没有合并到master，你执行 **git branch -d a** 是删除不了的，它会智能的提示你a分支还有未合并的代码，但是如果你非要删除，那就执行 **git branch -D a** 就可以强制删除a分支。

git tag

我们在客户端开发的时候经常有版本的概念，比如v1.0、v1.1之类的，不同的版本肯定对应不同的代码，所以我一般要给我们的代码加上标签，这样假设v1.1版本出了一个新bug，但是又不晓得v1.0是不是有这个bug，有了标签就可以顺利切换到v1.0的代码，重新打个包测试了。

所以如果想要新建一个标签很简单，比如 **git tag v1.0** 就代表我在当前代码状态下新建了一个v1.0的标签，输入 **git tag** 可以查看历史 tag 记录。



```
→ test git:(a) git tag
v1.0  / 新建图文消息
v1.1
```

可以看到我新建了两个标签 v1.0、v1.1。

想要切换到某个tag怎么办？也很简单，执行 **git checkout v1.0**，这样就顺利的切换到 v1.0 tag的代码状态了。

OK，以上全是一些最基本的Git操作，而且全是在本地环境进行操作的，完全没有涉及到远程仓库，下一章节将以远程 GitHub 仓库为例，讲解下本地如何跟远程仓库一起同步协作，另外今天讲的全是最基础最简单的Git操作，一步步来，后续再继续讲解一下Git的高阶以及一些Git的酷炫操作。

另外，考虑到可能会有人嫌我讲解的太基础太慢，毕竟我是针对小白，所以得一步步来，迫不及待的想要提前自己学习的不妨回复「git」关键字，获取一份我推荐的还不错的 Git 学习资料，不谢，毕竟我这么帅！