

# 从0开始学习 GitHub 系列之「Git 进阶」

原创 2016-06-16 stormzhang AndroidDeveloper



阅读本文大概需要 5 分钟。

关于 Git 相信大家看了之前一系列的文章已经初步会使用了，但是关于Git还有很多知识与技巧是你不知道的，今天就来给大家介绍一下一些 Git 进阶的知识。

## 1 / 用户名和邮箱

我们知道我们进行的每一次 commit 都会产生一条 log，这条 log 标记了提交人的姓名与邮箱，以便其他人方便的查看与联系提交人，所以我们在进行提交代码的第一步就是要设置自己的用户名与邮箱。执行以下代码：

```
git config --global user.name "stormzhang"  
git config --global user.email "stormzhang.dev@gmail.com"
```

以上进行了全局配置，当然有些时候我们的某一个项目想要用特定的邮箱，这个时候只需切换到你的项目目录，以上代码把 **--global** 参数去除，再重新执行一遍就ok了。

PS：我们在 GitHub 的每次提交理论上都会在主页的下面产生一条绿色小方块的记录，如果你确认你提交了，但是没有绿色方块显示，那肯定是你提交代码配置的邮箱跟你 GitHub 上的邮箱不一致，GitHub 上的邮箱可以到 **Setting -> Emails** 里查看。

## 2 / alias

我们知道我们执行的一些 Git 命令其实操作很频繁的类似有：

```
git commit  
git checkout  
git branch  
git status  
...
```

这些操作非常频繁，每次都要输入完全是不是有点麻烦，有没有一种简单的缩写输入呢？比如我想直接输入以下命令代替：

```
git c  
git co  
git br
```

**git s**

...

是不是很简单快捷啊？这个时候就用到了 **alias** 了，翻译过来就是别名的意思，输入以下命令就可以直接满足以上的需求。

**git config --global alias.co checkout # 别名**

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

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

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

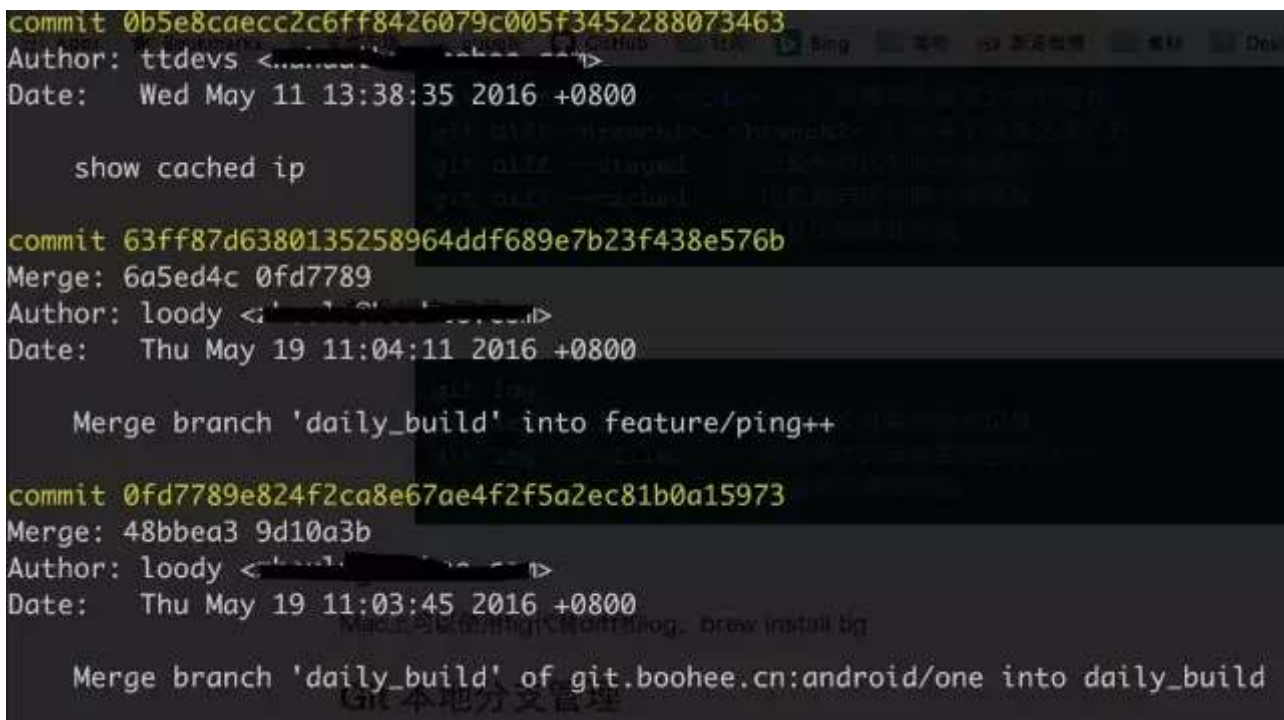
当然以上别名不是固定的，你完全可以根据自己的习惯去定制，除此之外还可以设置组合，比如：

**git config --global alias.psm 'push origin master'**

**git config --global alias.plm 'pull origin master'**

之后经常用到的 **git push origin master** 和 **git pull origin master** 直接就用 **git psm** 和 **git plm** 代替了，是不是很方便？

另外这里给大家推荐一个很强大的 **alias** 命令，我们知道我们输入 **git log** 查看日志的时候是类似这样的：



```
commit 0b5e8caeccc2c6ff8426079c005f3452288073463
Author: ttdevs <[redacted]>
Date:   Wed May 11 13:38:35 2016 +0800

    show cached ip

commit 63ff87d6380135258964ddf689e7b23f438e576b
Merge: 6a5ed4c 0fd7789
Author: loody <[redacted]>
Date:   Thu May 19 11:04:11 2016 +0800

    Merge branch 'daily_build' into feature/ping++

commit 0fd7789e824f2ca8e67ae4f2f5a2ec81b0a15973
Merge: 48bbea3 9d10a3b
Author: loody <[redacted]>
Date:   Thu May 19 11:03:45 2016 +0800

    Merge branch 'daily_build' of git.boohie.cn:android/one into daily_build
```

告诉大家一个比较屌的命令，输入

```
git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative
```

然后日志这样了：

```
* 0b5e8ca - (HEAD, feature/debugger) show cached ip (4 hours ago) <ttdevs>
* 63ff87d - (origin/s1, origin/feature/ping++, feature/ping++) Merge branch 'daily_build' into daily_build (4 weeks ago) <loody>
| \
| * 0fd7789 - Merge branch 'daily_build' of 'wanglinglong:feature/ping++' into daily_build (4 weeks ago) <loody>
| \
| * 9d10a3b - 修改饮食工具 (4 weeks ago) <wanglinglong>
| * 6a5ed4c - Merge branch 'daily_build' into feature/ping++ (4 weeks ago) <loody>
| \
| * 48bbea3 - Merge branch 'daily_build' of 'wanglinglong:feature/ping++' into daily_build (4 weeks ago) <loody>
| \
| * 2441d30 - 工具饮食改进 (4 weeks ago) <wanglinglong>
| * c7b8c38 - 修改评测 (4 weeks ago) <wanglinglong>
| * 6f4c304 - 评测适配小屏幕手机&优化 (4 weeks ago) <wanglinglong>
| * 8df50ef - 优化用户评测 (4 weeks ago) <wanglinglong>
| * 6b96942 - 处理BaseJsonRequest中判断是不是food逻辑错误 (4 weeks ago) <ttdevs>
| * 4a9649f - 用户在开始时设置的目标时间已到不需要再评测 (4 weeks ago) <wanglinglong>
| * 6297df5 - 个人资料界面调整 (4 weeks ago) <wanglinglong>
| * f493984 - 刻度尺精度 (4 weeks ago) <wanglinglong>
| * c1a5489 - 修改评测相关问题 (5 weeks ago) <wanglinglong>
| * ca5320d - Merge branch 'daily_build' of 'wanglinglong:feature/ping++' into daily_build (5 weeks ago) <loody>
| \
| * 34c17fe - (origin/feature/init) 完善评测 (5 weeks ago) <wanglinglong>
| * 1237856 - 修改用户评测 (5 weeks ago) <wanglinglong>
| * 6bd2e71 - Merge branch 'master' into daily_build (5 weeks ago) <loody>
```

是不是比较清晰，整个分支的走向也很明确，但是每次都要输这么一大串是不是也很烦？这时候你就该想到 **alias** 啊：

```
git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative"
```

这样以后直接输入 **git lg** 就行了。

## 3 / 其他配置

当然还有一些其他有用的配置，默认情况下 git 用的编辑器是 vi，如果不喜欢可以改成其他编辑器，比如我习惯 vim。

```
git config --global core.editor "vim" # 设置Editor使用vim
```

你们如果喜欢其他编辑器可自行搜索配置，前提是本机有安装。

有些人纳闷我的终端怎么有各种颜色显示，自己却不是这样的，那是因为你们没有开启给 Git 输出着色，输入如下命令即可：

```
git config --global color.ui true
```

还有些其他的配置如：

```
git config --global core.quotePath false # 设置显示中文文件名
```

以上的配置基本就差不多了，默认这些配置都在 `~/.gitconfig` 文件下的，你可以找到这个文件查看自己的配置，也可以输入 **git config -l** 命令查看。

## 4 / diff

**diff** 命令算是很常用的，使用场景是我们经常在做代码改动，但是有的时候2天前的代码了，做了哪些改动都忘记了，在提交之前需要确认下，这个时候就可以用diff来查看你到底做了哪些改动，举个例子，比如我有一个 a.md 的文件，我现在做了一些改动，然后输入 **git diff** 就会看到如下：



```
diff --git a/a.md b/a.md
index ea8f022..d2a5109 100644
--- a/a.md
+++ b/a.md
@@ -1,3 @@
-aaaaaaa
+bbbbbbb
+ccccccc
+ddddddd
```

红色的部分前面有个 - 代表我删除的，绿色的部分前面有个 + 代表我增加的，所以从这里你们能一目了然的知道我到底对这个文件做了哪些改动。

值得一提的是直接输入 **git diff** 只能比较当前文件和缓存区文件差异，什么是缓存区？就是你还没有执行 **git add** 的文件。

当然跟暂存区做比较之外，他还可以有其他用法，如比较两次 commit 之间的差异，比较两个分支之间的差异，比较缓存区和版本库之间的差异等，具体用法如下：

```
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
```

## 5/ checkout

我们知道 **checkout** 一般用作切换分支使用，比如切换到 develop 分支，可以执行：

```
git checkout develop
```

但是 **checkout** 不只用作切换分支，他可以用来切换 tag，切换到某次 commit，如：

```
git checkout v1.0
git checkout ffd9f2dd68f1eb21d36cee50dbdd504e95d9c8f7
```

# 后面的一长串是commit\_id，是每次commit的SHA1值，可以根据 git log 看到。

除了有“切换”的意思，**checkout** 还有一个撤销的作用，举个例子，假设我们在一个分支开发一个小功能，刚写完一半，这时候需求变了，而且是大变化，之前写的代码完全用不了了，好在你刚写，甚至都没有 **git add** 进暂存区，这个时候很简单的一个操作就直接把原文件还原：

## git checkout a.md

这里稍微提下，**checkout** 命令只能撤销还没有 add 进暂存区的文件。

# 6 / stash

设想一个场景，假设我们正在一个新的分支做新的功能，这个时候突然有一个紧急的bug需要修复，而且修复完之后需要立即发布。当然你说我先把刚写的一点代码进行提交不就行了么？这样理论上当然是ok的，但是这会产品垃圾commit，原则上我们每次的commit都要有实际的意义，你的代码只是刚写了一半，还没有什么实际的意义是不建议就这样commit的，那么有没有一种比较好的办法，可以让我暂时切到别的分支，修复完bug再切回来，而且代码也能保留的呢？

这个时候 **stash** 命令就大有用处了，前提是我们的代码没有进行 commit，哪怕你执行了 add 也没关系，我们先执行

## git stash

什么意思呢？就是把当前分支所有没有 commit 的代码先暂存起来，这个时候你再执行 **git status** 你会发现当前分支很干净，几乎看不到任何改动，你的代码改动也看不见了，但其实是暂存起来了。执行

## git stash list

你会发现此时暂存区已经有了一条记录。

这个时候你可以切换回其他分支，赶紧把bug修复好，然后发布。之后一切都解决了，你再切换回来继续做你之前没做完的功能，但是之前的代码怎么还原呢？

## git stash apply

你会发现你之前的代码全部又回来了，就好像一切都没发生过一样，紧接着你最好需要把暂存区的这次 stash 记录删除，执行：

## git stash drop

就把最近一条的 stash 记录删除了，是不是很方便？其实还有更方便的，你可以使用：

## git stash pop

来代替 apply 命令，pop 跟 apply 的唯一区别就是 pop 不但会帮你把代码还原，还自动帮你把这条 stash 记录删除，省的自己再 drop 一次了，虽然更方便，但是使用起来也需要更加谨慎，为了验证你可以紧接着执行 **git stash list** 命令来确认是不是已经没有该记录了。

最后还有一个命令介绍下：

## git stash clear

就是清空所有暂存区的记录，**drop** 是只删除一条，当然后面可以跟 stash\_id 参数来删除指定的某条记录，不跟参数就是删除最近的，而 **clear** 是清空。

# 7/ merge & rebase

我们知道 **merge** 分支是合并的意思，我们在一个 featureA 分支开发完了一个功能，这个时候需要合并到主分支 master 上去，我们只需要进行如下操作：

**git checkout master**

**git merge featureA**

其实 **rebase** 命令也是合并的意思，上面的需求我们一样可以如下操作：

**git checkout master**

**git rebase featureA**





以上截图里就是冲突的示例，冲突的地方由 ===== 分出了上下两个部分，上部分一个有 **HEAD** 的字样代表是我当前所在分支的代码，下半部分是一个叫 **baidu\_activity** 分支的代码，可以看到 **HEAD** 对 gradle 插件进行了升级，同时新增了一个插件，所以我们很容易判断哪些代码该保留，哪些代码该删除，我们只需要移除掉那些老旧代码，而且同时也要把那些 <<< **HEAD**、===== 以及 >>> **baidu\_activity** 这些标记符号也一并删除，最后进行一次 commit 就ok了。

我们在开发的过程中一般都会约定尽量大家写的代码不要彼此影响，以减少出现冲突的可能，但是冲突总归无法避免的，我们需要了解并掌握解决冲突的方法。



[点击阅读原文报名野狗技术沙龙](#)

PS：昨天的文章是软文，因为客户要求不让开评论，今天这里说一下，虽然是软文，但是极客学院绝对是正规的，相信不少人也看过他们网站的视频，大家自己根据自身情况以及需要去判断选择就好。

相关文章：

[「从0开始学习 GitHub 系列之「向GitHub 提交代码」」](#)

[「从0开始学习 GitHub 系列之「Git速成」」](#)