使用 git status -s 命令或 git status --short 命令，你将得到一种格式更为紧凑的输出。新添加的未跟踪文件前面有 ?? 标记，新添加到暂存区中的文件前面有 A 标记，修改过的文件前面有 M 标记。

要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。 可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

git log --oneline –graph 当 oneline 或 format 与另一个 log 选项 --graph 结合使用时尤其有用。 这个选项添加了一些 ASCII 字符串来形象地展示你的分支、合并历史：

你提交后发现忘记了暂存某些需要的修改，可以像下面这样操作：

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

最终你只会有一个提交——第二次提交将代替第一次提交的结果。

**撤消对文件的修改**

如果你并不想保留对 CONTRIBUTING.md 文件的修改怎么办？ 你该如何方便地撤消修改——将它还原成上次提交时的样子（或者刚克隆完的样子，或者刚把它放入工作目录时的样子）？ 幸运的是，git status 也告诉了你应该如何做。 在最后一个例子中，未暂存区域是这样：

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

它非常清楚地告诉了你如何撤消之前所做的修改。 让我们来按照提示执行：

$ git checkout -- CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

可以看到那些修改已经被撤消了。

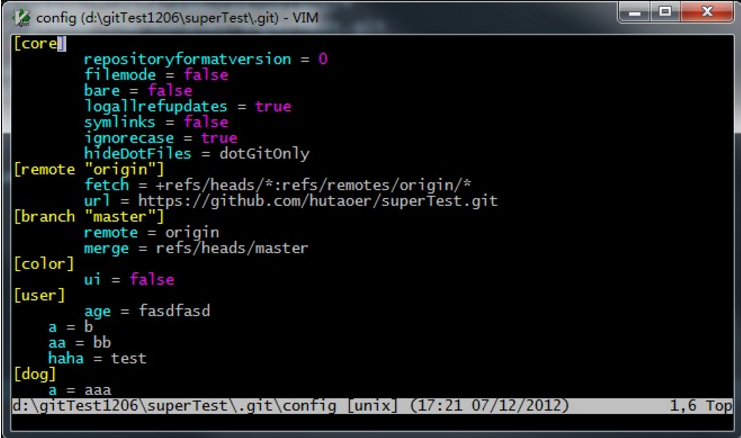
平时我们在使用git的时候，很少去关注其配置是如何，而在实际开发中，对git config这个命令的使用也并不是很多，但是配置对一个程序和项目来说都是很重要的，我们今天来看看git的配置以及git config的初步应用。

    1. git config简介

我们知道config是配置的意思，那么git config命令就是对git进行一些配置。而配置一般都是写在配置文件里面，那么git的配置文件在哪里呢？互动一下，先问下大家。

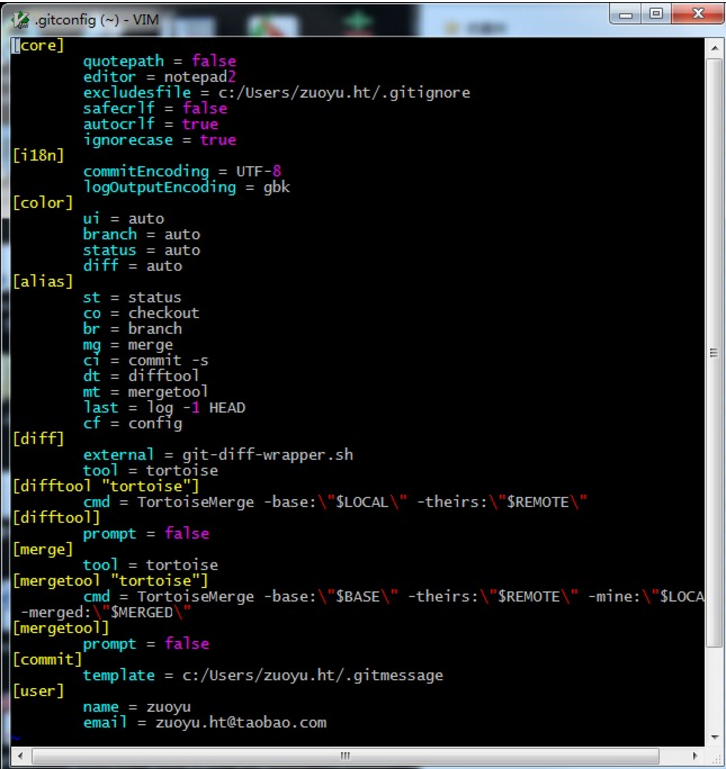
你们所知的git配置文件是放在哪里的？git有几个配置文件呢？ 是的，聪明的你，稍微查查资料就知道咯，git里面一共有3个配置文件，首先是： 仓库级配置文

件：该文件位于当前仓库下，路径.git/，文件名为.gitconfig，这个配置中的设置只对当前所在仓库有效 仓库级配置文件内容如下：

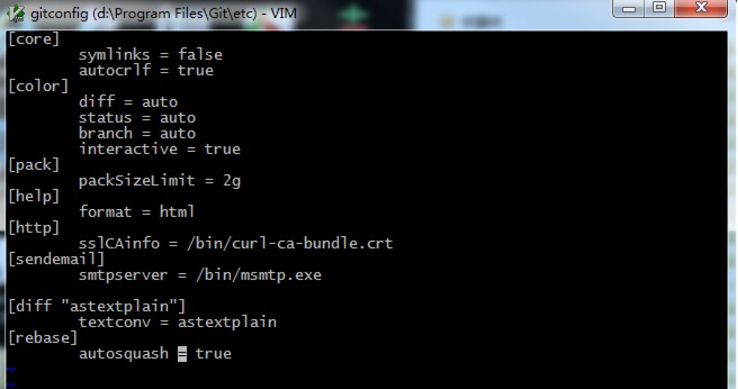


第二个是 全局级配置文件：win7下面路径，在用户目录下，以个人的PC机为例，其路径为：C:\Users\zuoyu.ht，文件名为.gitconfig

全局级配置文件内容如下：



最后是系统级配置文件：本地git的安装目录下，以左御的git安装路径为例：D:\Program Files\Git\etc，文件名为：gitconfig，内容如下：



从三个图的对比中，我们可以发现，有些配置项的名称是一样的，比如“core”，在git中，每一项的配置名称又叫做“section”，

“section”下面的每一行对应着一个key和一个value。需要注意的是：系统级配置文件不同于另外两个配置文件，是因为他少了

一个点，因此用vim打开的时候，是没有高亮显示的。”core”这个section在每个配置文件中都有，那么它git最终会调用哪个配置

文件呢？这就要有一个先后顺序了。

2. 配置文件如何生效的

用一个我们大家都熟悉的例子来比较，就是css。一般来说，在一个页面中引入css文件的方法有三种，见： http://www.cnblogs.com/hutaoer/archive/2010/12/07/1899105.html ，

通俗的讲就是行内样式“<div style=“width:200px”>”，内嵌样式“<style type=“text/css”>div{width:200px}</style>”

和外部样式“<link rel=“stylesheet” href=“div.css”>”。这三种样式的权重是行内>内嵌>外部。类似的，对于git来说，配置文件

的权重是仓库>全局>系统。 Git会使用这一系列的配置文件来存储你定义的偏好，它首先会查找/etc/gitconfig文件（系统级），该

文件含有对系统上所有用户及他们所拥有的仓库都生效的配置值。接下来Git会查找每个用户的~/.gitconfig文件（全局级）。最后

Git会查找由用户定义的各个库中Git目录下的配置文件.git/config（仓库级），该文件中的值只对当前所属仓库有效。以上阐述的三

层配置从一般到特殊层层推进，如果定义的值有冲突，以后面层中定义的为准，例如：.git/config和/etc/gitconfig的较量中，

.git/config取得了胜利。虽然你也可以直接手动编辑这些配置文件，但是运行git config命令将会来得简单些。下面我们就来看看如

何利用git config命令来编辑配置各个级别的文件。

    3. 用git config命令查看配置文件

命令参数 –list, 简写 -l

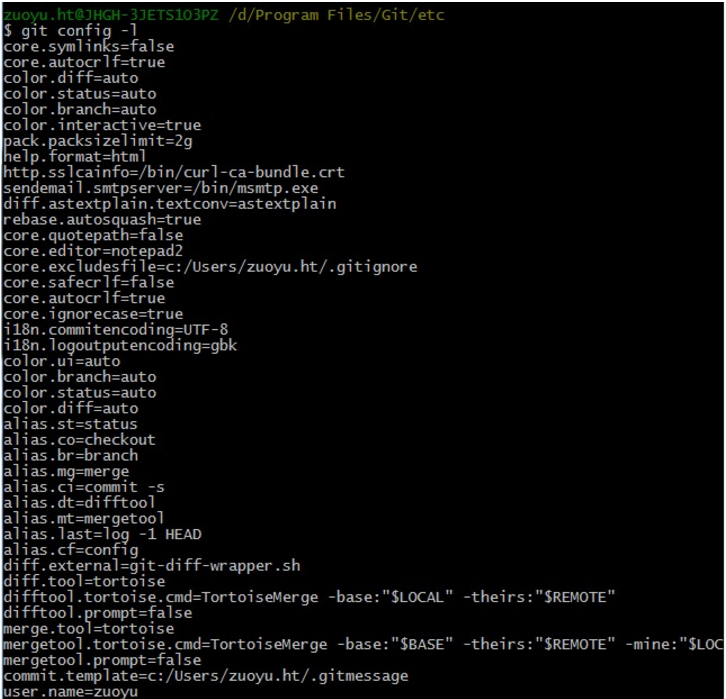
格式：git config [–local|–global|–system] -l

查看仓库级的config，即.git/.config，命令：git config –local -l

查看全局级的config，即C:\Users\zuoyu.ht\.gitconfig，命令：git config –global -l

查看系统级的config，即D:\Program Files\Git\etc\gitconfig，命令：git config –system -l

查看当前生效的配置，命令：git config -l，这个时候会显示最终三个配置文件计算后的配置信息，如下图：



    4. 使用git config命令编辑配置文件

编辑的英文单词是什么，没错，edit

命令参数 –edit, 简写 -e

格式：git config [–local|–global|–system] -e

查看仓库级的config，即.git/.config，命令：git config –local -e，与–list参数不同的是，git config -e默认是编辑仓库级的配置文件。

查看全局级的config，即C:\Users\zuoyu.ht\.gitconfig，命令：git config –global -e

查看系统级的config，即D:\Program Files\Git\etc\gitconfig，命令：git config –system -e

   执行这个命令的时候，git会用配置文件中设定的编辑器打开配置文件。

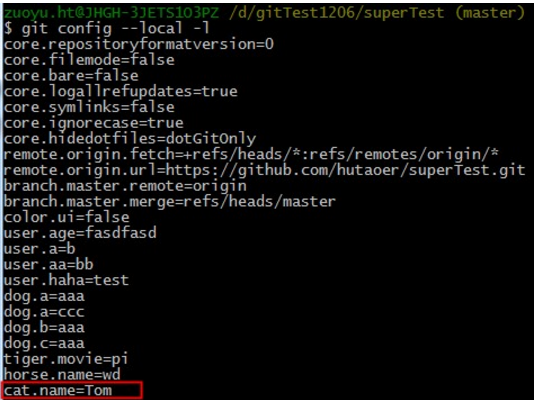
      5. 增加一个配置项

参数 –add

格式: git config [–local|–global|–system] –add section.key value(默认是添加在local配置中)

注意add后面的section,key,value一项都不能少，否则添加失败。比如我们执行：git config –add cat.name tom

然后查看local中的配置：



注意增加一项配置而不赋值 git config –add cat.age，或者单单增加一个section,git config –add cat1，不会成功。

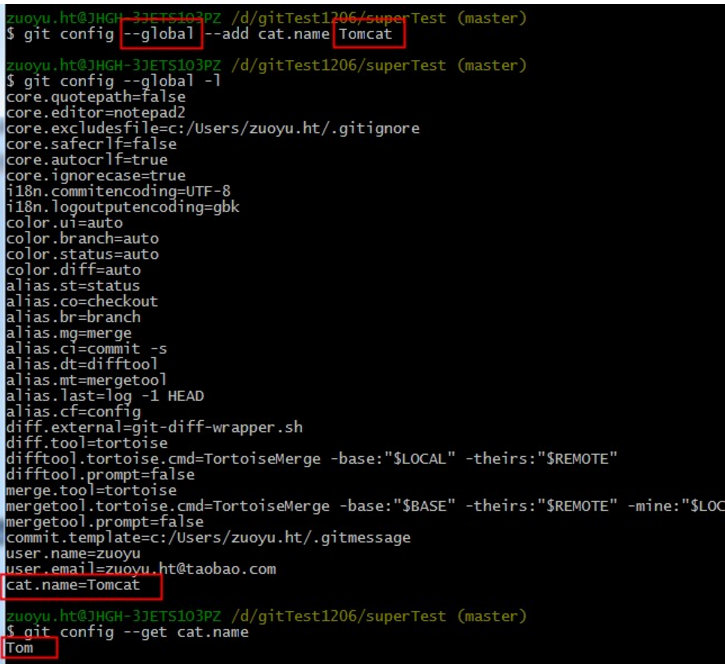
6.获取一个配置项

有时候，我们并不需要查看所有配置的值，而是查看某个配置项的值，怎么做呢？

命令参数 –get

格式：git config [–local|–global|–system] –get section.key(默认是获取local配置中内容)

我们先往global配置中写入一个cat.name=Tomcat的配置项，再使用git config –get cat.name看看得到的是什么



结果就是local中的cat.name=Tom，因此git config –get section.key 等价于git config –local –get section.key

如果获取一个section不存在的key值，不会返回任何值

如果获取一个不存在的section的key值，则会报错

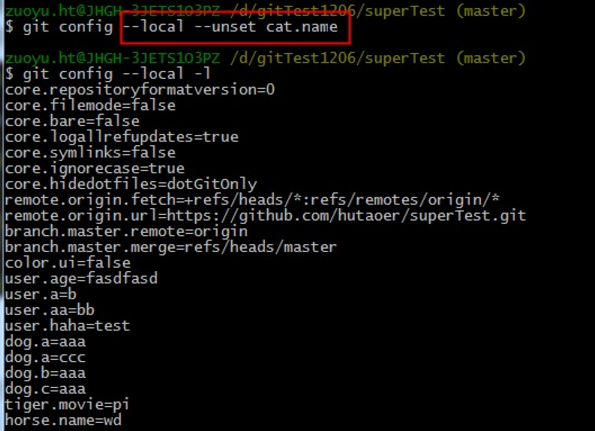


  7.删除一个配置项

命令参数 –unset

格式：git config [–local|–global|–system] –unset section.key

相信有了前两个命令的使用基础，大家举一反三就知道改怎么用了，来，我们试试删除local配置中的cat.name



可见，我们添加的cat.name已经被删除了

但是，在配置文件中，cat这个section还依然存在，那么怎么通过命令来彻底删除呢？

**git config命令使用第二篇——section操作，多个key值操作，使用正则**

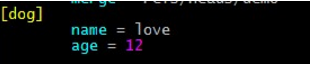
1. 删除一个section

命令参数 --remove-section

格式：git config [--local|--global|--system] --remove-section section

使用这个命令，不仅可以删除一个没有内容的section，而且即使该section下面有内容，也会一起删除掉

比如我们来删除这样一个配置:





直接使用命令: git config --local --remove-section dog，这样可以把dog下面的内容全部删除，而且也能删除掉section

2. 查看value类型

一共可以查看四种类型：--bool, --int, --bool-or-int, --path

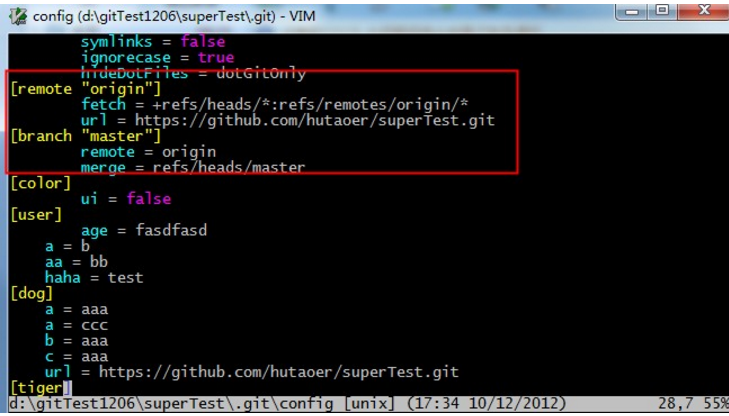
格式：git config [--local|--global|--system] [--bool|--int|--bool-or-int|--path] section.key

使用示例：git config --local --bool dog.name

如果dog.name不是布尔值，且也不能被转化为bool值，那么git就会报错；否则git会显示true或false

3. 操作特殊的section

看看这类section长什么样子的



这类section后面还跟了一个字段，例如[remote "origin"]

对这类section应该如何使用git config进行操作呢？很简单把section后面的字段也作为section的key写进去，这样会多一层嵌套，但是忽略了这一层就会报错了

比如我们要对[remote "master"]下的url做修改，git config --local remote.origin.url value，value为修改后的值，这样就ok啦，而增删改查都是一样的

4. 重命名section

参数：--rename-section

格式：git config [--local|--global|--system] --rename-section section1 section2

比如我们把名为dog的section改为dog1, git config --local rename-section dog dog1，这样就好咯

5. 替换，获取和删除多个属性

参数：--replace-all, --get-all, -unset-all

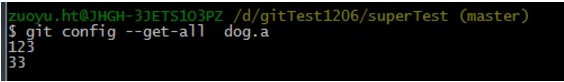
如果此时，配置中有,dog.a=dddd,同时还有dog.a = aaaa,如下图



然后，可以对这些进行修改, git config --local --replace-all dog.a 333

这时候dog.a值均被替换，且只保留最后一个，这个是需要注意的

获取配置中所有dog.a的value值



删除配置中所有dog.a的value值



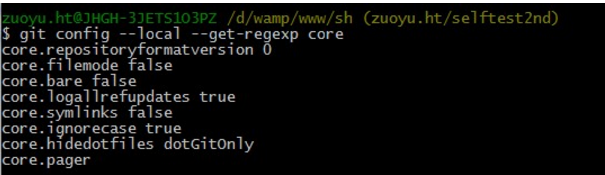
6. 运用正则初步

参数： --replace-all, --get-regexp

正则很强大，同样的，在git中也很强大，git中并没有专门的获取某个section下面所有key值value的方法，但是试用--get-regexp这个参数就可以实现

例如我们来获取配置中的core下面的所有key值的value

使用命令：get config --local --get-regexp core 即可



--replace-all中也可以使用正则的

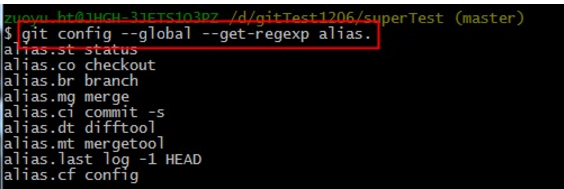
git config第三篇——git的常用配置

1. 设置别名

section:alias

比如我们来设置config的缩写命令为cf，其命令如下：git config alias.cf config

看看一般的我们都设置了些什么样的别名，这让我们在操作git的时候，事半功倍哦。



2. 设置编辑器

section.key <=> core.editor

git会默认调用你当前配置定义的editor作为文本编辑器，如果没有定义的话，会调用vim来创建和编辑提交以及标签信息

设置vim为默认的编辑器 git config core.editor vim

git其实已经内置了vim作为默认的编辑器，在git的安装目录下，可以找到vim的exe文件

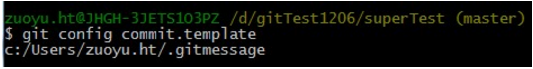
如果配置其他的编辑器的话，需要指定编辑器的调用路径才行

3. 设置默认提交文案

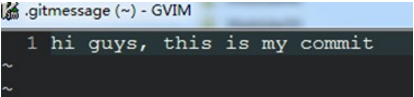
section.key <=> commit.template

如果把此项指定为系统上的一个文件，当你提交的时候，Git会默认使用该文件定义的内容

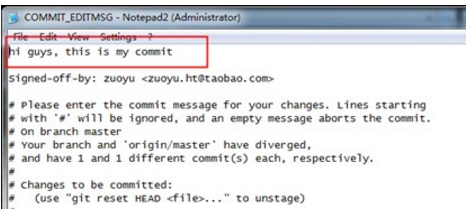
比如我指定了用户目录下的.gitmessage文件



创建message：“hi guys, this is my commit”



最终在提交commit的时候显示如下：



4. 设置分页

section.key <=> core.paper [less|more|''] 默认为less

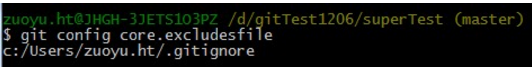
设置为less的时候，如果内容超过了限制就需要使用分页器，按space空格键可以下翻页

设置为more或空的时候，会把内容全部显示

5. 设置忽略文件和目录

和.gitignore文件具有类似的功能，用来过滤不需要被跟踪的文件和目录

类似于commit.template，我们也可以指定一个.gitignore文件



在.gitignore中写入不需要被跟踪的文件和目录

6. 工具配置

可以配置difftool，mergetool等等，帮助我们更好的使用git

具体的配置细节请参考《Pro Git》，下载地址：http://download.csdn.net/detail/hutaoer06051/4871069

7. 格式化空白

section.key <=> core.autocrlf

在windows系统下，使用回车和换行两个字符来结束一行，而Mac和Linux下只使用一个字符。因此在不同操作系统的下的协作方式也不同。

**在git操作中，我们经常会用到fetch, merge, pull和push等命令，以下是一些我们需要注意的地方。**

给大家准备了参考资料：

1. Whatʼs a Fast Forward Merge?：https://sandofsky.com/images/fast\_forward.pdf

2. Understanding the Git Workflow：https://sandofsky.com/blog/git-workflow.html

3. Understanding Git: Merging：http://www.sbf5.com/~cduan/technical/git/git-3.shtml

$ git pull origin test

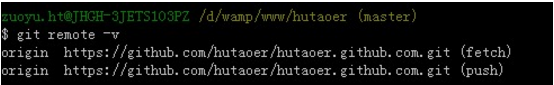
// git pull合并代码的时候，若发生冲突，会处于merging状态，检查代码，发现自己的分支低于主分支，这个时候想撤销merge

// 撤销merge

$ git reset --hard HEAD (or sha\_1)

git pull = git fetch + git merge

在git仓库中运行命令：git remote -v

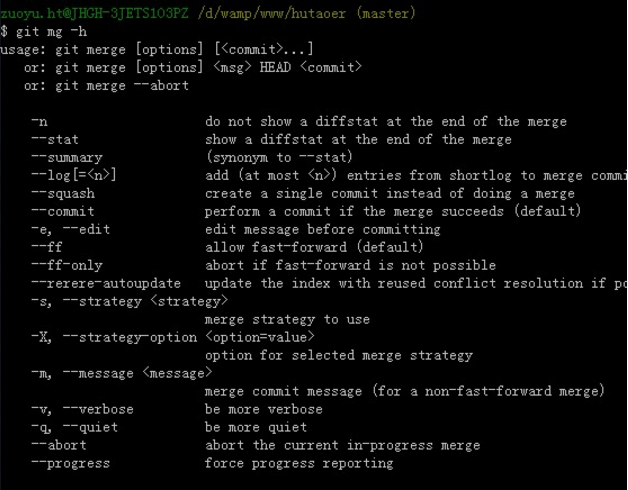


可以看到fetch和push命令可以分别对远程分支进行fetch和push操作，而pull不是直接跟远程分支对话的。

fetch同pull的区别在于：git fetch:是从远程获取最新版本到本地，不会自动merge 而git pull是从远程获取最新版本并merge到本地仓库 从安全角度出发，git fetch比git pull更安全，因为我们可以先比较本地与远程的区别后，选择性的合并。 git push 默认推送到master，如果有多个分支，则多个分支一起推送到远程

git merge

git merge其实有很多参数的哦，我们来看看merge有哪些参数可以供我们选择:

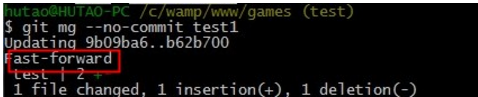


git merge -n <branch> 不会在合并后显示合并前后的不同状态

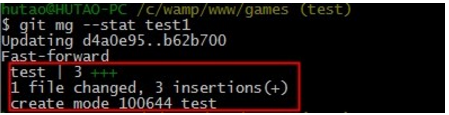
git merge –stat <branch> 跟上面的参数相反，它会在合并结束后显示合并前后的不同状态

git merge -e <branch> 在合并之前会调用配置的编辑器，您可以自己编写commit，否则的话commit的内容就是git自动生成的

git merge –no-commit <branch>相当于一次快进式合并，不会生成一次提交，如下图所示：



git merge –stat <branch> 在合并后会显示不同的状态，如下图红色框中部分：



git merge –no-stat <branch> 合并后不会显示不同的状态，如下图：



git merge –quiet <branch> 无声的合并（不会输出任何信息）

git merge –ff <branch> 当合并是快进式合并的时候，仅仅是更新了分支的指针，不会产生合并提交，这也是默认的合并行为

git merge –no-ff <branch> 及时是快进式合并，也会创建一个合并提交

git merge –ff-only <branch> 只允许快进式合并

快进式合并和非快进式合并

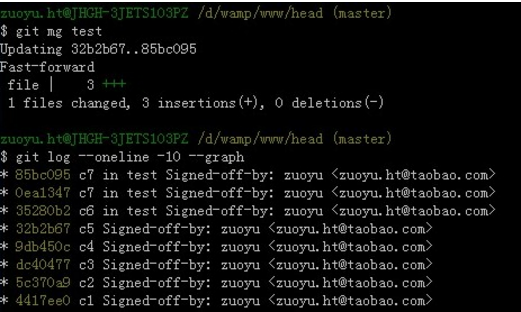
何为快进式合并？？

如果当前的分支和另一个分支没有内容上的差异，就是说当前分支的每一个提交(commit)都已经存在另一个分支里了，git 就会执行一个“快速向前”(fast forward)操作；git 不创建任何新的提交(commit),只是将当前分支指向合并进来的分支。

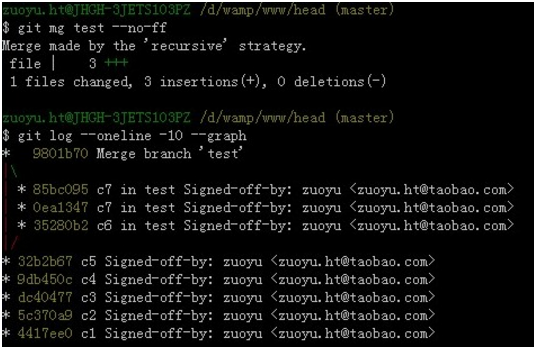
快进式和非快进式的区别：

目前仓库的状态是，master分支上有了5次提交，然后基于master创建分支test，并在test上进行3次提交，接下来，我在master分支上对test分支分别进行快进式和非快进式合并。

先进行快进式合并， git merge test (默认为–ff合并)，如下图所示，我们看到一共是5+3=8个提交，分支的提交历史也没有“开叉”，即没有多个父提交节点。



随后，我回到master的第五次提交，然后对test进行非快进式合并，git merge –no-ff test



这时候，同快进式合并不同的是，在c5处产生了“开叉”，然后新生成了一个commit “Merge branch 'test'”，即多了一个父提交。

现在，大家应该对这两种合并有了一种直观的了解了吧。

非快进式合并

非快进式合并会生成新的提交，并“开叉”，可以让我们的提交历史更加的清晰！

http://www.cnblogs.com/hutaoer/archive/2013/03/14/3078869.html

转载于:https://www.cnblogs.com/softidea/p/4967616.html