

## Contents

|  |    |
|--|----|
| 一、自述 .....   | 2  |
| 二、白话概念.....  | 3  |
| 三、我使用的命令.....  | 4  |
| 四、Git 的基本使用 1—直接在主分支上操作，无需 merge，无其他人修改主分支 .....               | 5  |
| 五、Git 基本使用 2—重复上面的步骤 .....                                     | 8  |
| 六、Git 的基本使用 3:---新建工作分支，做改动，然后 merge，模拟有人修改 master,无冲突         | 10 |
| 七、Git 的基本使用 4:工作分支改动，然后 merge，模拟改 master,有冲突（同一文件同一行）<br>..... | 15 |
| 八、Git 的基本使用 5:工作分支改动，然后 merge，模拟改 master,有冲突（同一文件不同行）<br>..... | 18 |
| 九、Git 的基本使用 6:---谁动了我的 master? .....                           | 19 |
| 十、现在你能分析整个过程吗？ .....   | 22 |

Doc name: 《git:菜鸟教菜鸟》

Author: Armand Wang

Date: July 30, 2013

E-mail: ww\_zidongkong@sina.com

Any advice rendered would be greatly appreciated. Thank you!

版本控制的最大好处就是让你永远可以后悔

----某大牛

## 一、自述

Git 工具结束了 ‘小作坊’ 式的生产，我们不在使用 copy 来保存各个版本，特别是大工程协作开发时，这种方式也是不可行的。每一次 commit 产生的 snapshot 都可以迅速保存当前的版本状态，它使用了如此精妙的算法和数据结构可以像 ‘照照片’ 一样快速的记录细微或庞大的改动，你可以通过 git log 展示的 tag，回到任何一个你想回到版本。大家都在对 master 分支上的 code 进行修改，贡献自己的力量，人与人的想法不同，当修改了同一个地方的时候，就会有冲突。这样就需要你 merge，你对？他对？或者都保留？这取决于你。**恢复版本、分布控制（包括冲突解决）、分支方案，是我认为 git 的三大特色。**

Git 是个工具，我们不必要知道它的实现，也不必要了解太多概念，经过运用后形成的感性认识是最重要的，**所以我决定写一些自己的直观感受和一己之见，绝非技术博客，倒可以是新手的入门文章，我不保证措辞标准，但是看下去，会用没问题。**

我也是刚学 git，只会一些简单的命令，但目前已经满足了我的工作需要，学习状态大致经历了三个阶段：

1，开始的时候完全不明白，只知道 source code 在远端，可以通过 git clone down 下来，可以通过 git reset --hard 恢复最初的状态。

2，知道了 git status/add/commit/push/pull 等基本操作，可以糊里糊涂的 push 和 pull。但是对于仓库，分支的概念很不清晰，也不会解决冲突。这一阶段多亏了实验室一个同学的帮助和 github 的教程。

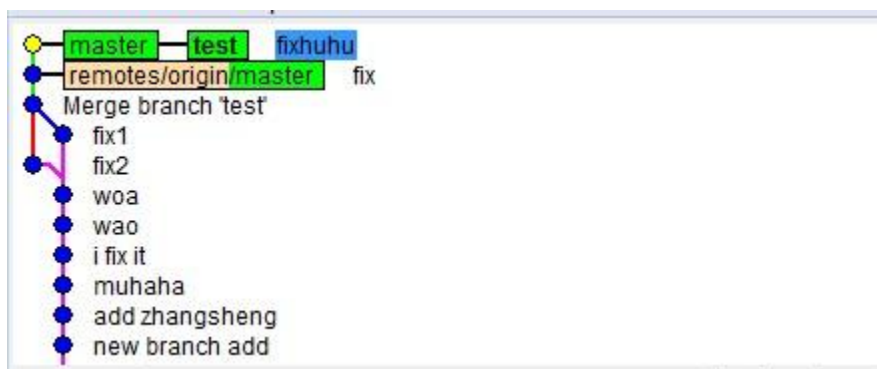
3，相当于柳暗花明的阶段，是读了一篇博文，知道了 add 在干嘛，commit 又是干嘛，最重要学习了一条指令，gitk -all，这是 git 的 GUI，上面的 code line 让我彻底明白了分支和 commit 的概念，当我**使用 git checkout 切换分支，发现仓库的内容竟然悄无声息的发生了改变**。我瞬间明白了到底怎么回事。每个分支都有自己的成长记录（git log），我可以通过 git checkout +ID 回到该分支的任何历史状态，此时你打开本地的仓库文件夹，它呈现的就是你所切换到的分支，你所要求的状态。在 GUI 上一切都用 code line 的方式体现出来，我的我的小伙伴们简直都惊呆了。

首先，推荐一个代码的公共托管平台 github，这是世界知名的以 git 为基础的编程服务中心，是很多程序员的最爱，你可以注册账号，就可以在上面建立 server 端的仓库，然后 clone 到本地，管理自己的代码。网址是：<https://github.com/> 既然是公共平台，我也没有放置重要的代码在上面，所以我的账号可以给你们用。账户名称：armand-wang 密码是 weide\_332020865。我已经新建了一个仓库，名字叫 learngit，你们可以使用

git clone [git@github.com:armand-wang/learngit.git](https://github.com:armand-wang/learngit.git)  
把仓库 clone 到本地。

## 二、白话概念

仓库是工程代码（开发包）的**容器**，**分支是仓库的‘代言人’**，你所看到的仓库永远是它的其中的某个分支，**分支可以看做版本**！每个版本都有自己的 code line，（可以看做成长记录，可以用 git log 查看）即每次 commit 之后都会生成一个快照，像拍照一样，形成一个版本号 tag，也相当于做了一次备份，**一步一个脚印...就这样，你的 code line 在生长，你的分支（版本）在不断进化**。神奇的是，你可以根据 tag 号，使用 git checkout 回到该版本的任意一个状态。比如：



这是 gitk -all 生成的本地仓库 code line，每一次 commit 都会生成一个节点，我们可以看到，节点显示的信息就是 git commit -m 后面字符串的内容，即提交的原因。图中所示：

①一开始只有 master 一个分支（软件版本），后来由于 ‘new branch add’ 的原因提交了第一次修改（即做了点成果，然后备份），再后来由于 ‘add zhangsheng’ 的原因，提交了第二次修改，相当于我又备份了一次，继续往前走，继续开发，由于 muhaha 的原因，我提交了第三次修改，好玩，备份后，继续开发，一直修改到了 woa 后的版本。

②这是我新建了一个分支 test，相当于克隆了一个自己，开始的时候完全一样，我切换到 test 分支，发现跟 master 分支的 woa 之后的版本是一样的，**即在一个分支上新建分支属于继承关系（说白了就是拷贝），这样本地仓库就有了两个‘代言人’。你想看谁，就切换到谁**。然后我在 test 分支（版本）上做了 ‘fix1’ 修改，在主分支上做了 ‘fix2’ 修改，我现在想将两次修改合并，即两个版本生成一个版本，master 分支下使用 git merge test，这样 master 分支就具有了两个版本的特性，当然如果两次修改了同一个文件，就要求你根据提示信息处理冲突。

③合并之后我对 master 做了一次名为 fix 的修改，然后 push 到了 remote 仓库的 master 分支上。在 push 之前所有的操作都是 local 的。然后对 master 做了一次 fixhuhu 修改。

④最后在 test 分支下，使用个 git merge master 将 test 分支 merge 到主分支上，这样 test 和 master 两个版本又回到了同一个节点。

上面的过程是一个典型的过程，但是我们一般不在主分支上做改动，我只是为了模拟其他人改动了主分支。我们新建一个分支如 test 分支，做自己的改动，最后回到主分支，pull 下最新的代码（别人 commit 后 push 的）然后将自己的分支 merge 到主分支上，解决冲突后 push 到远端的 master 分支。

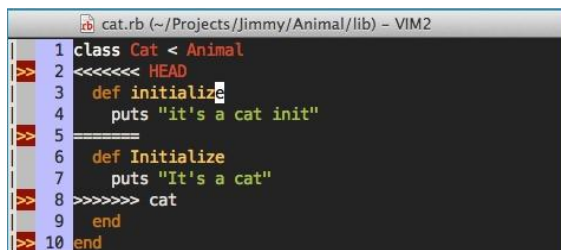
上图中，我在 git log 或者 gitk -all 中都可以查看到每个节点的 tag，即版本 ID，我可以通过 git checkout ID 回到这个节点时的状态，git branch 会提示你不在任何分支上。Gitk -all 可以看到你所在的节点的颜色是与其他节点不同的。

### 三、我使用的命令

我现在所使用的**都是一些比较简单又非常非常重要的命令**，但能满足正常的 push/pull 等需求。后面我将对这些命令进行使用介绍，先熟悉一下：

|                      |   |
|----------------------|---|
| git clone +repo 地址   | 即将仓库下载到本地，并置于 git 的监控（追踪）之下                 |
| git rest -hard       | 使仓库中的代码版本恢复到最近的一个节点。                        |
| Git branch           | 查看本地的分支情况及当前分支                              |
| Git branch + 新分支名称   | 新建分支  |
| Git checkout +分支名    | 切换分支  |
| Git status           | 查看当前状态，它的提示信息很有用                            |
| git add .            | 将所有的修改置于 git 的监控之下，提交之前必须要做的。               |
| Git commit -m 'info' | 提交一次修改，生成节点,在本地仓库做一次备份，必须填写 info，即修改的原因和细节。 |
| Git pull             | 从远端获取更新                                     |
| Git merge +分支        | 分支合并，即将两个版本合并成一个版本。                         |
| git push             | （没有冲突后）将 local 版本 push 到远端。                 |
| Git branch -D +分支    | 删除分支  |
| Git checkout +ID     | 恢复到某个节点                                     |
| Git log              | 查看当前分支的成长记录                                 |
| Gitk -all            | 显示 GUI，这里的 code line 在初学的时候特别有用。            |

要养成经常使用 git status 和 gitk -all 的习惯。新建分支命令到 push 命令，命令时依次用到，完成一次 down 代码，修改代码，上传代码的过程。出现冲突后，根据提示打开冲突文件会有下面的画面：



## Git: 菜鸟教菜鸟—by Armand

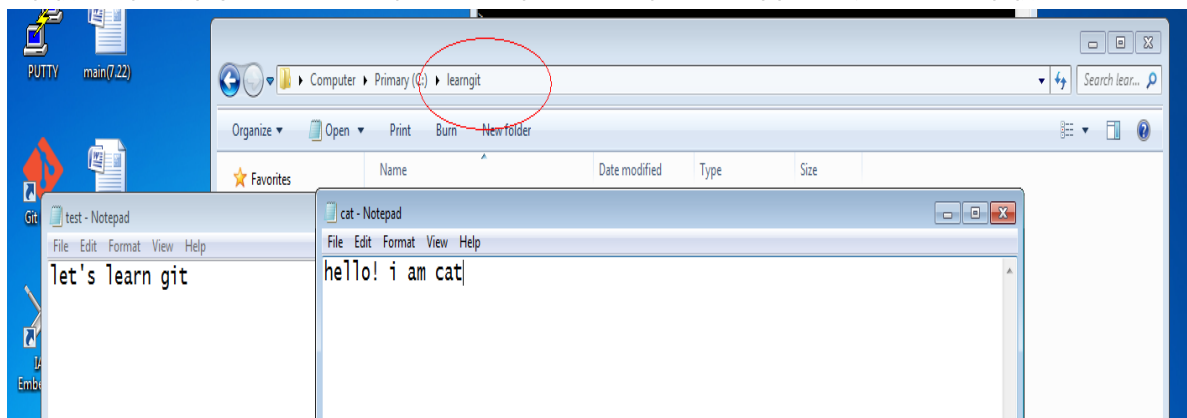
两行箭头之间用=好隔开两部分，我们选择其一或者都留下，记得一定要删除这些提示符，只保存内容。依次解决冲突之后，我们再次执行 `git add` 和 `git commit`。就完成了合并。这是会在 `code line` 上生成一个节点，也能很明显看到这个节点时分支合并的汇聚点。

### 四、Git 的基本使用 1—直接在主分支上操作，无需 merge，无其他人修改主分支

在阐述之前，清理一下本地仓库，删除除 `master` 之外的所有分支，清理 `master` 版本的文件内容，以备后续的教程。趁这个机会，第一次演示基本步骤使用 `git branch` 发现有两个分支，删除 `test1`

```
B46395@B46395-01 /c/learngit (master)
$ git branch
* master
  test1
B46395@B46395-01 /c/learngit (master)
$ git branch -D test1
Deleted branch test1 (was f78ddf7).
```

如图所示，仓库中有两个文件，`test.txt` 和 `cat.txt`，文件内容被我修改到了如图所示。



使用 `git status` 之后发现：文件被修改的提示。而且提示我们把修改添加到提交内容

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
# (use "git push" to publish your local commits)
#
# 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:   cat.txt
#       modified:   test.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
B46395@B46395-01 /c/learngit (master)
```

## Git: 菜鸟教菜鸟—by Armand

我们使用 `git add` .使被修改的文件由 `unstage` 状态转到 `stage` 状态，你可以想象成是把这些修改放到了工作台上，准备提交了。

```
$ git add .
B46395@B46395-01 /c/learnGit (master)
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   cat.txt
#       modified:   test.txt
B46395@B46395-01 /c/learnGit (master)
$
```

下面我们提交一次修改，并将修改的原因写为 `begin study`，如下：

```
B46395@B46395-01 /c/learnGit (master)
$ git commit -m 'begin study'
[master 81497af] begin study
2 files changed, 2 insertions(+), 23 deletions(-)
B46395@B46395-01 /c/learnGit (master)
$
```

我们利用 `gitk --all` 查看 `codeline`

The screenshot shows the `gitk` GUI. The left pane displays the commit history graph with the `master` branch highlighted in green. The right pane shows the commit log for the `master` branch, listing commits by `armand-wang` with timestamps. The bottom pane shows the commit details for the selected commit, including the author, committer, parent, branch, and the commit message `begin study`.

我们发现，`master` 所在的节点，最新的 commit 是 `begin study`.这个节点是我们学习的开始。这时我们 `git status` 一下：

```
B46395@B46395-01 /c/learnGit (master)
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
B46395@B46395-01 /c/learnGit (master)
$
```



这时显示提交过两个备份，即 code line 中显示的 data 和 begin study 两次提交。既然到这一步我们不妨将此时的 master push 到远端。在 push 之前我们先 pull 一下，看有没有人修改过 master 的代码，如果有修改过就牵涉到合并、解决冲突的问题，如果没有修改过就万事大吉。当然除了我，目前还没有人操作我的仓库。所以：

```
$ git pull
Already up-to-date.
B46395@B46395-01 /c/learngit (master)
$
```

即显示没有更新。此时我们可以 push 了。

```
$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

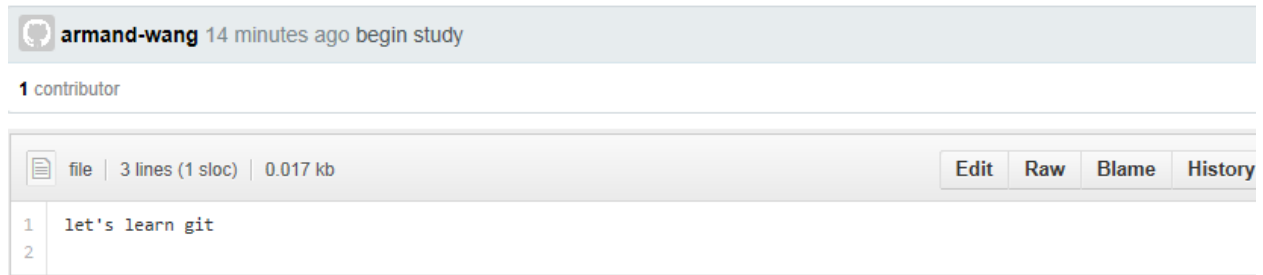
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 594 bytes, done.
Total 7 (delta 2), reused 0 (delta 0)
To git@github.com:armand-wang/learngit.git
   f78ddf7..81497af master -> master
B46395@B46395-01 /c/learngit (master)
$
```

我一般都是使用 git push，不加参数，它会直接 push 到默认的仓库分支。To [git@github.com:armand-wang/learngit.git](https://github.com/armand-wang/learngit.git)

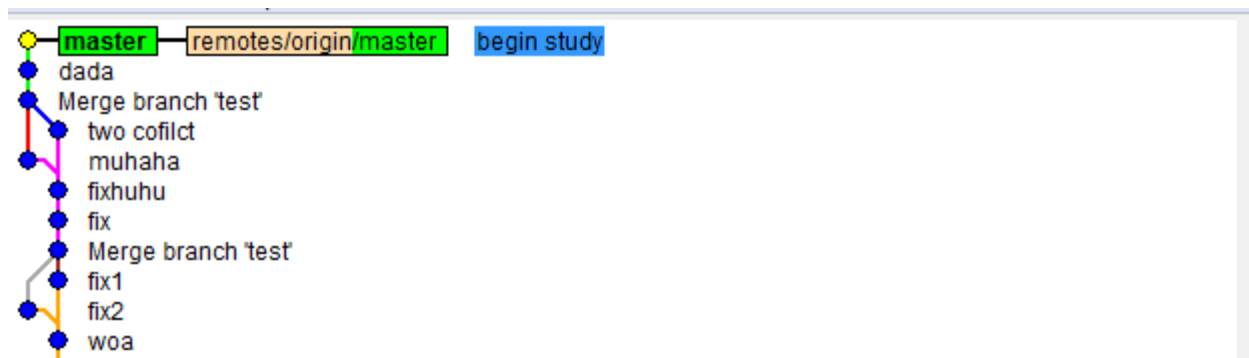
我们登录到 github 上查看 push 的结果：



# Git: 菜鸟教菜鸟—by Armand

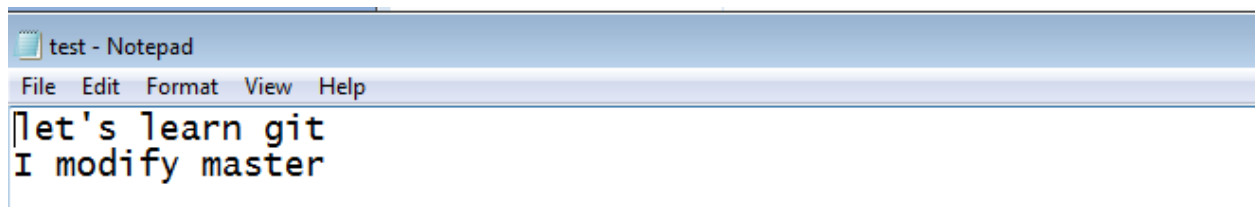


即文档已经正确上传到了远端仓库。现在的 code line 如下：



## 五、Git 基本使用 2—重复上面的步骤

上面的步骤，是在 master 分支直接操作，push 之前，无需 merge，而且没有别的人修改主分支。为了强调这个步骤的重要性，我将再次重复这个过程，在主分支上修改 test.txt。



Git status

```
$ git status
# On branch master
# 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:   test.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
846395@846395-01 /c/learngit (master)
```

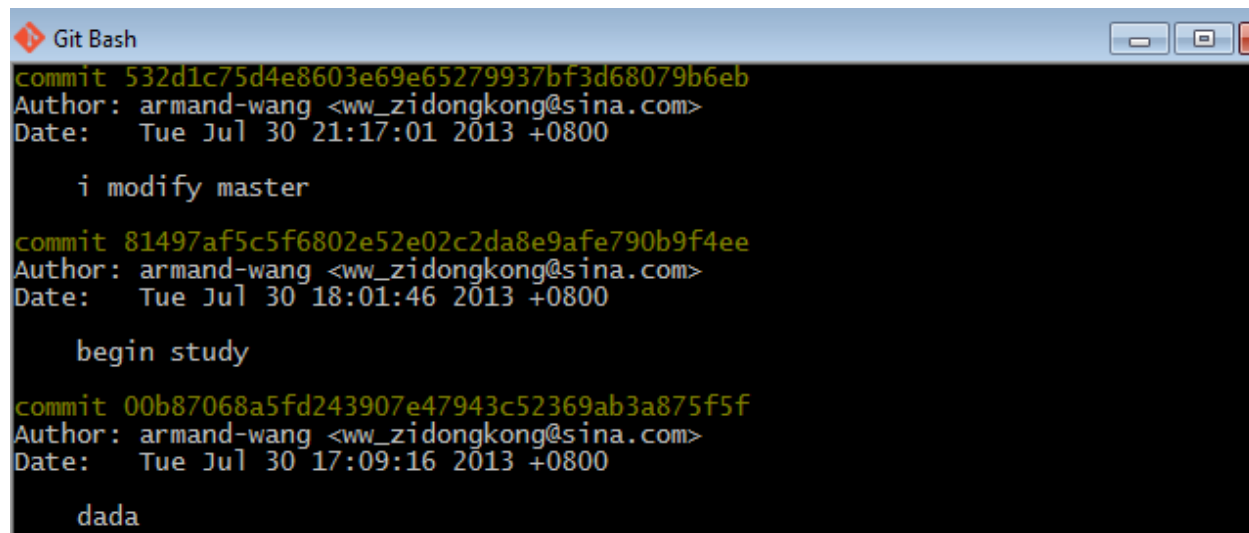
Git add . && git commit -m 'I modify master'



## Git: 菜鸟教菜鸟—by Armand

```
$ git add
Nothing specified, nothing added.
Maybe you wanted to say 'git add .'?
846395@846395-01 /c/learngit (master)
$ git add .
846395@846395-01 /c/learngit (master)
$ git commit -m 'i modify master'
[master 532d1c7] i modify master
1 file changed, 1 insertion(+)
846395@846395-01 /c/learngit (master)
```

修改已经提交了，我们通过 `git log` 可以查看日志



```
commit 532d1c75d4e8603e69e65279937bf3d68079b6eb
Author: armand-wang <ww_zidongkong@sina.com>
Date:   Tue Jul 30 21:17:01 2013 +0800

    i modify master

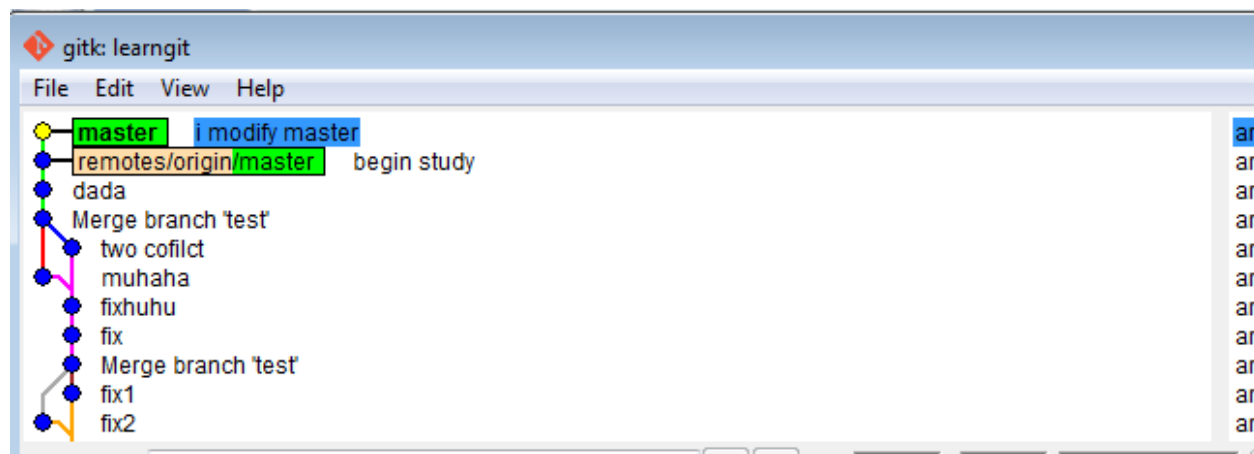
commit 81497af5c5f6802e52e02c2da8e9afe790b9f4ee
Author: armand-wang <ww_zidongkong@sina.com>
Date:   Tue Jul 30 18:01:46 2013 +0800

    begin study

commit 00b87068a5fd243907e47943c52369ab3a875f5f
Author: armand-wang <ww_zidongkong@sina.com>
Date:   Tue Jul 30 17:09:16 2013 +0800

    dada
```

通过 `gitk -all` 可以看到 code line 上也显示了此次提交形成的节点



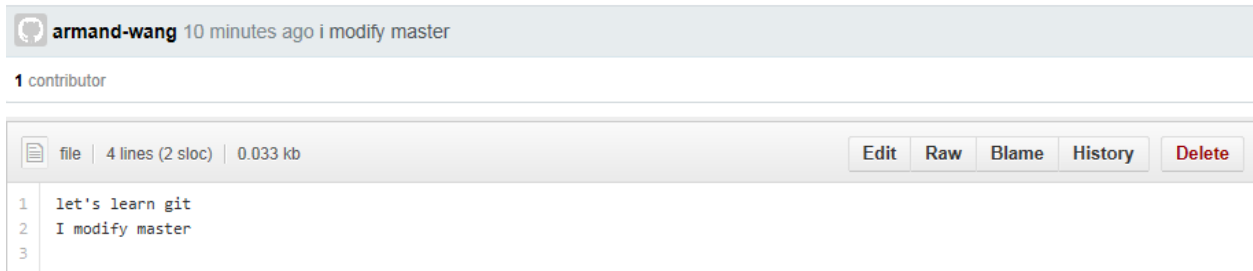
此时我们将这个修改提交到 remote(因为我确信没有人修改过主分支，所以省去了 pull)

Git push

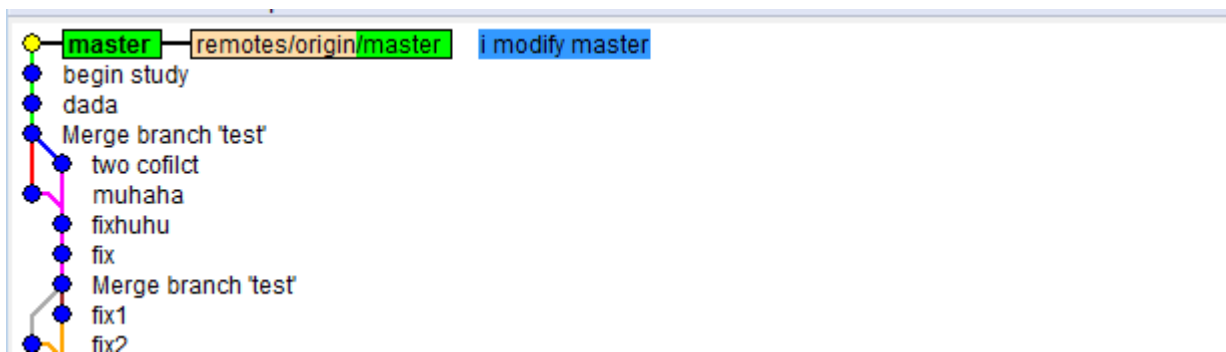
## Git: 菜鸟教菜鸟—by Armand

```
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.com:armand-wang/learngit.git
81497af..532d1c7 master -> master
B46395@B46395-01 /c/learngit (master)
$
```

我们查看 github，已经成功修改。



Gitk -all，也显示了这次的 push



### 六、Git 的基本使用 3:---新建工作分支，做改动，然后 merge，模拟有人修改 master,无冲突

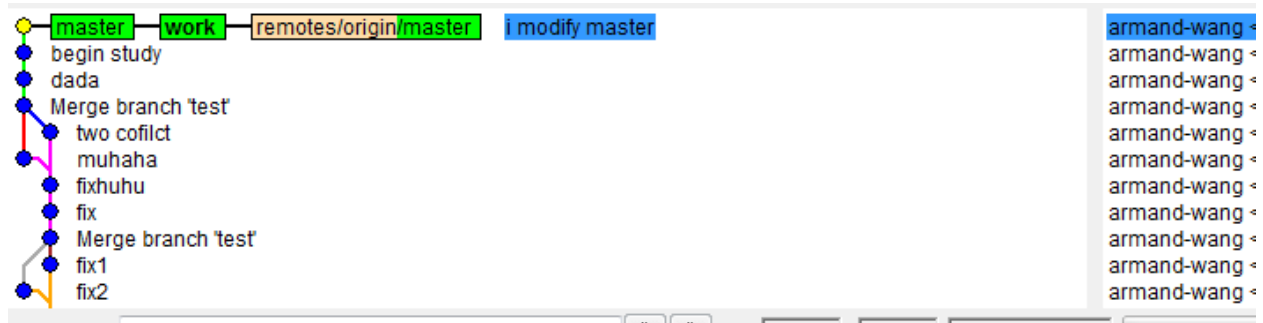
现在，我们已经知道了基本的步骤，下面我们练习添加工作分支，在工作分支做修改，然后与主分支合并（merge），先不涉及有冲突的情况。我们首先在 master 上新建工作分支 work,建成之后 work 与 master 是继承关系（拷贝）所以它们在同一个节点（版本）。

Git branch work 然后切换到这个分支。

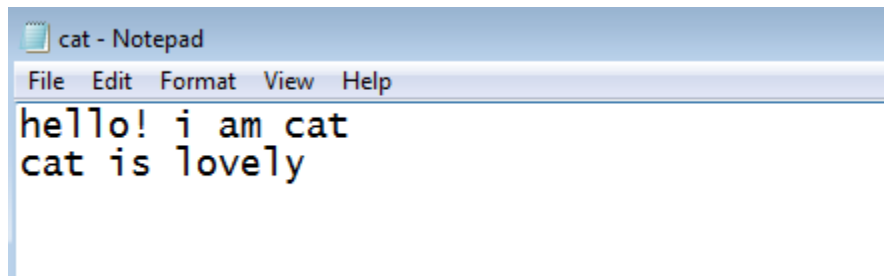
```
B46395@B46395-01 /c/learngit (master)
$ git branch work
B46395@B46395-01 /c/learngit (master)
$ git checkout work
Switched to branch 'work'
B46395@B46395-01 /c/learngit (work)
$
```

Gitk -all，我们可以看到 work 与 master 的关系。

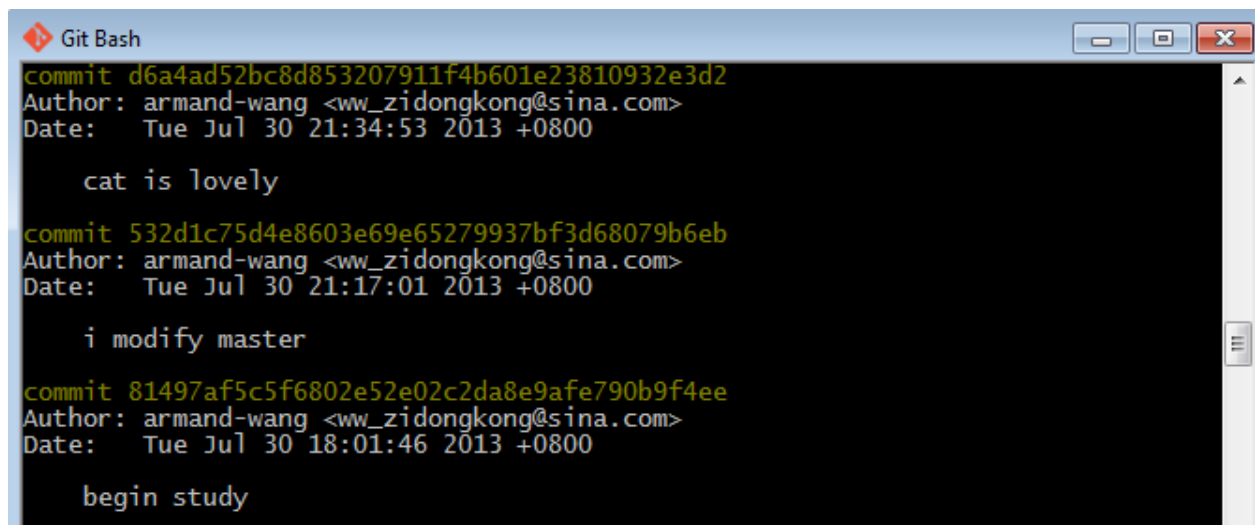
## Git: 菜鸟教菜鸟—by Armand



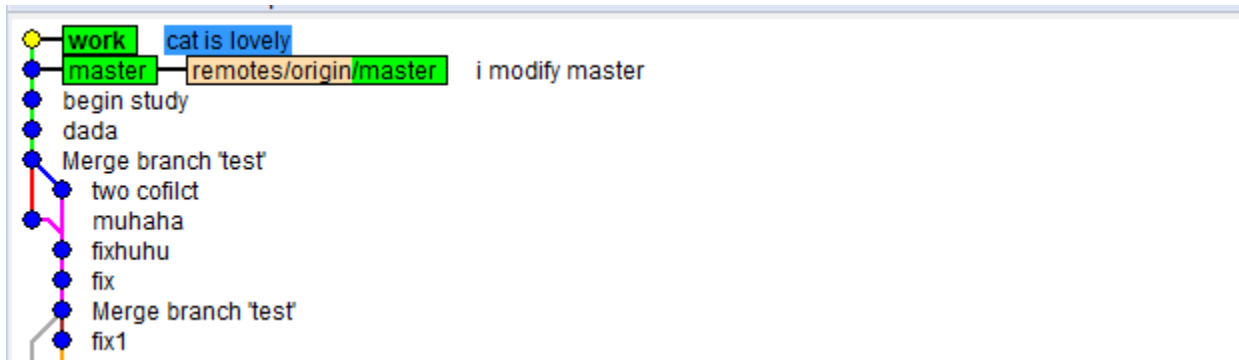
在 work 分支下，我们修改 cat.text 文件，在其中添加一样：cat is lovely



重复之前的步骤提交这个修改，原因是 cat is lovely, Git status/git add ./git commit -m 'cat is lovely' 最后查看 work 分支的 log 和仓库的 code line 如下：



## Git: 菜鸟教菜鸟—by Armand



下面我们在模拟有人修改了 master 分支。在主分支增添了 mouse.txt 文件

先切换到主分支，再添加 mouse.txt 文件，内容为，hello, I am mouse

```
B46395@B46395-01 /c/learnGit (work)
$ git checkout master
Switched to branch 'master'
B46395@B46395-01 /c/learnGit (master)
$
```

然后提交这次修改，修改的信息时 add mouse,最后查看主分支的 log 和仓库 code line

```
Git Bash
commit ad59a880f1288476ce8b267448ca879b04aed369
Author: armand-wang <ww_zidongkong@sina.com>
Date: Tue Jul 30 21:44:33 2013 +0800

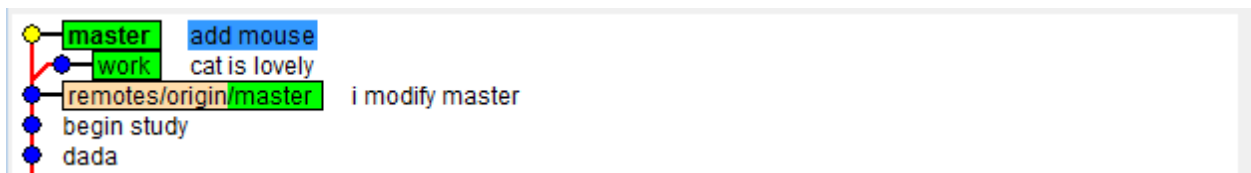
    add mouse

commit 532d1c75d4e8603e69e65279937bf3d68079b6eb
Author: armand-wang <ww_zidongkong@sina.com>
Date: Tue Jul 30 21:17:01 2013 +0800

    i modify master

commit 81497af5c5f6802e52e02c2da8e9afe790b9f4ee
Author: armand-wang <ww_zidongkong@sina.com>
Date: Tue Jul 30 18:01:46 2013 +0800

    begin study
```









记住我们模拟的情形是，work 是工作分支,在我们工作的时候 master 被以 add mouse 的原因修改了。而且已经被我们从远端的 master 分支上 pull 下来了。我们要将 cat is lovely 的修改也提

---

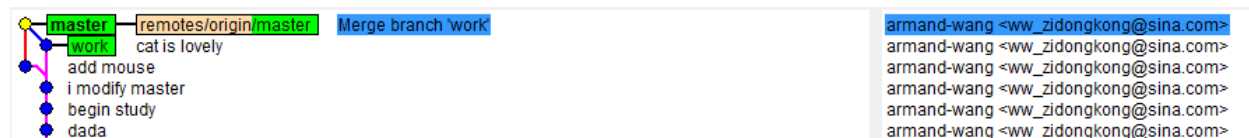
### Git push

```
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 740 bytes, done.
Total 8 (delta 3), reused 0 (delta 0)
To git@github.com:armand-wang/learngit.git
  532d1c7..5e6178b master -> master
B46395@B46395-01 /c/learngit (master)
$
```

查看 remote 即 github，修改已经成功上传

| Merge branch 'work'   |                                    |                          |   |
|---|------------------------------------|--------------------------|---|
|  | armand-wang authored 5 minutes ago | latest commit 5e6178bcd8 |  |
|  | <a href="#">README.md</a>          | Initial commit           | 19 days ago   |
|  | <a href="#">cat.txt</a>            | cat is lovely            | 19 minutes ago  |
|  | <a href="#">mouse.txt</a>          | add mouse                | 10 minutes ago  |
|  | <a href="#">test.txt</a>           | i modify master          | 37 minutes ago  |

Gitk -all 上面也记录了此次 push 的信息



看这个 code line，因为是一步一步看过来的，所以分析起来也容易，在 ‘I modify master’ 之后的版本，生出了 work 分支，然后 master ‘add mouse’，work 添加了 ‘cat is lovely’，然后主分支 merge 了 work，然后 push 到了远端。

为了让工作分支也保持最新的特性，我们切换到工作分支，让工作分支 merge 主分支，是他们再次站到同一个节点上。

```
B46395@B46395-01 /c/learngit (master)
$ git checkout work
Switched to branch 'work'
B46395@B46395-01 /c/learngit (work)
$ git merge master
Updating d6a4ad5..5e6178b
Fast-forward
 mouse.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 mouse.txt
B46395@B46395-01 /c/learngit (work)
$
```

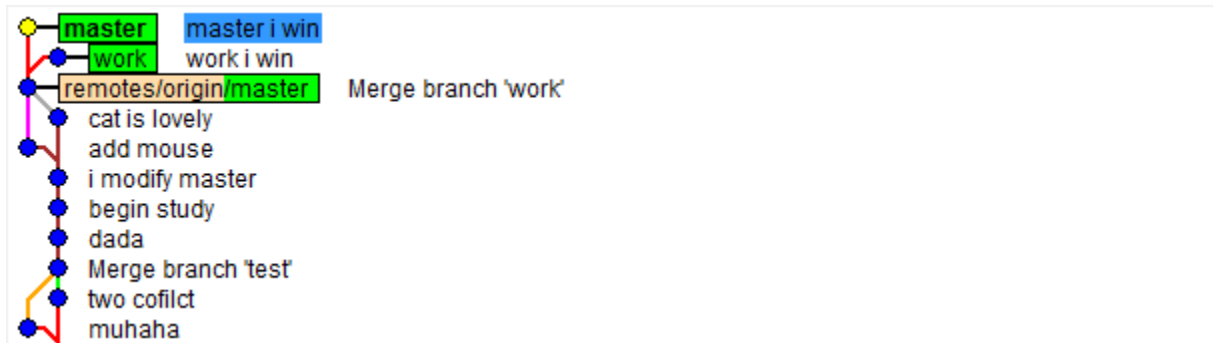


此时的 code line 变得如我们所料



### 七、Git 的基本使用 4:工作分支改动，然后 merge，模拟改 master,有冲突（同一文件同一行）

我们计划在 work 分支中的 cat.txt 最后一行添加 'work I win'，commit 信息也提交这一句。在 master 分支中在 cat.txt 的最后一行添加 'master I win'，commit 信息也提交这一句，修改主分支同样是模拟其他程序员修改了远端主分支，我们 push 之前 pull 下来到本地主分支。这两步都做完后，code line 如下：



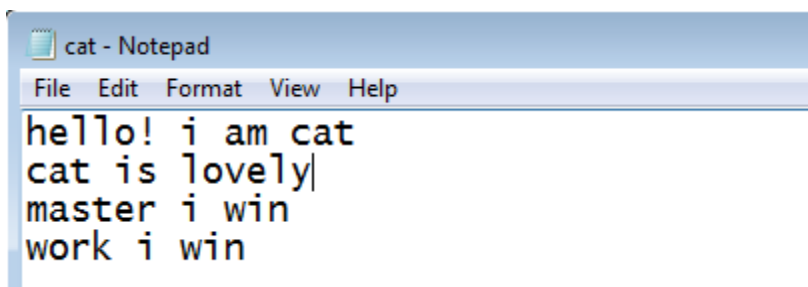
我们修改了同一个文件的同一行，我想 work 分支 merge 到主分支上是会出现冲突：

```
B46395@B46395-01 /c/learnGit (master)
$ git merge work
Auto-merging cat.txt
CONFLICT (content): Merge conflict in cat.txt
Automatic merge failed; fix conflicts and then commit the result.
B46395@B46395-01 /c/learnGit (master|MERGING)
$
```

我们此时查看 cat.txt 时，情况如下：

```
hello! i am cat
cat is lovely
<<<<<<< HEAD
master i win
=====
work i win
>>>>>>> work
```

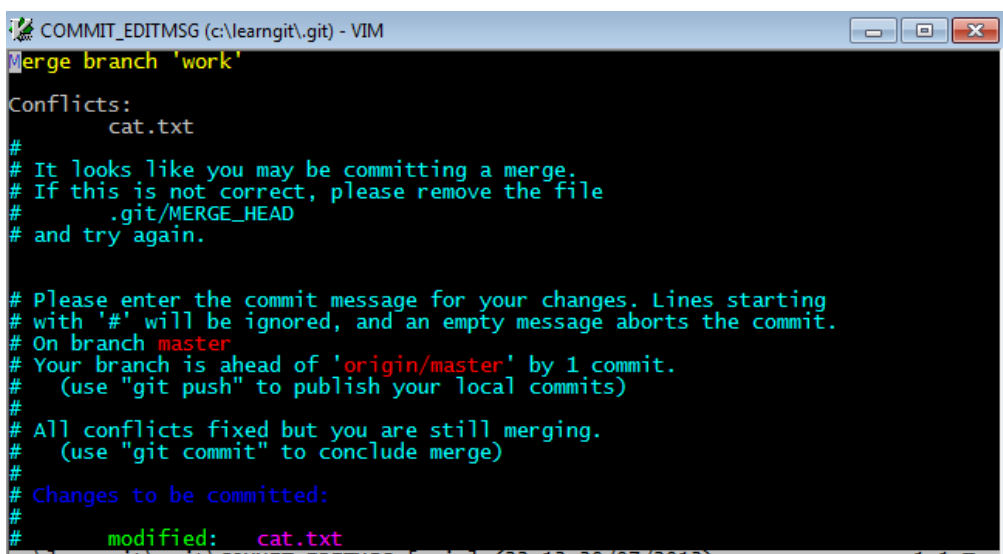
它显示了两个分支在这个地方的不同，我们选择其一，或者两个都保留，我们选择两个都保留，一定要记得删除所有的提示符。



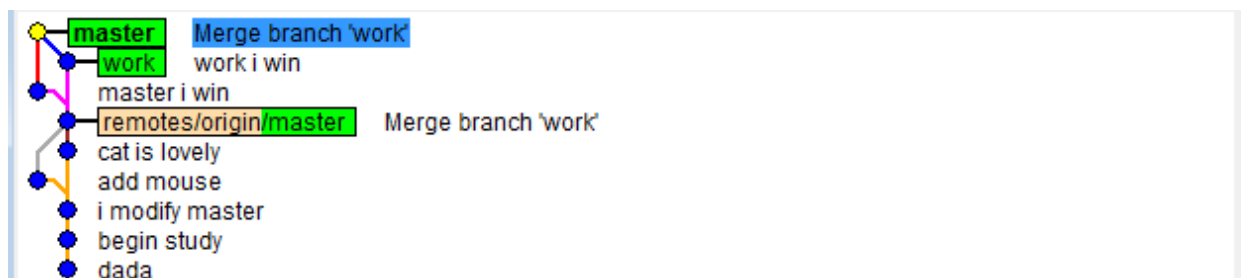
此时我们再次 git add.

```
B46395@B46395-01 /c/learnGit (master|MERGING)
$ git add .
B46395@B46395-01 /c/learnGit (master|MERGING)
$
```

Git commit 不加任何参数，出现下面的画面，直接：wq 退出，merge 成功



此时我们查看 code line




这个开始看起来十分复杂的 code line 你心里心里也很清楚每一步是怎么来的了吧？

下面可以 push 了

```
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (9/9), 830 bytes, done.
Total 9 (delta 4), reused 0 (delta 0)
To git@github.com:armand-wang/learngit.git
   5e6178b..c1bda46  master -> master
B46395@B46395-01 /c/learngit (master)
$
```

查看远端仓库中的 cat.txt 修改成功

 **armand-wang** 5 minutes ago Merge branch 'work'

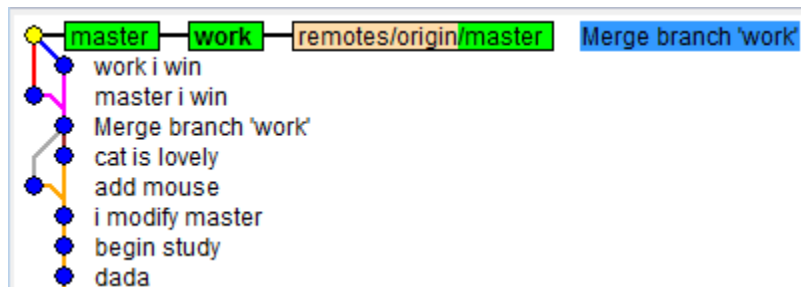
1 contributor

file | 5 lines (4 sloc) | 0.054 kb

|   |                 |
|---|-----------------|
| 1 | hello! i am cat |
| 2 | cat is lovely   |
| 3 | master i win    |
| 4 | work i win      |

同样，为了保持 work 分支中也拥有最新的特性，我在 work 分支中将 master 分支也 merge 进来，最后的 code line 你心中有数了吗？

如下：



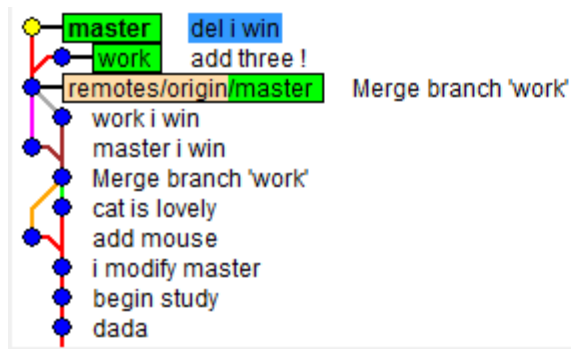
### 八、Git 的基本使用 5:工作分支改动，然后 merge，模拟改 master,有冲突（同一文件不同行）

经过前面的几个基本使用方法，我们看到工作分支与主分支修改的不是同一文件时，merge 过程中不会出现冲突，但是如果修改了同一文件的同一行，就会出现冲突，那么如果修改同一文件的不同行呢？会不会有冲突？答案我也不知道，下面就一起试试吧。

在 work 分支中修改 cat 的第一行 hello! i am cat 后面追加三个感叹号。commit info 是 add three !

在 master 分支中，修改 cat 的最后一行 work i win，删去 I win.commit 信息时 del I win

分别提交修改后 code line 变成了



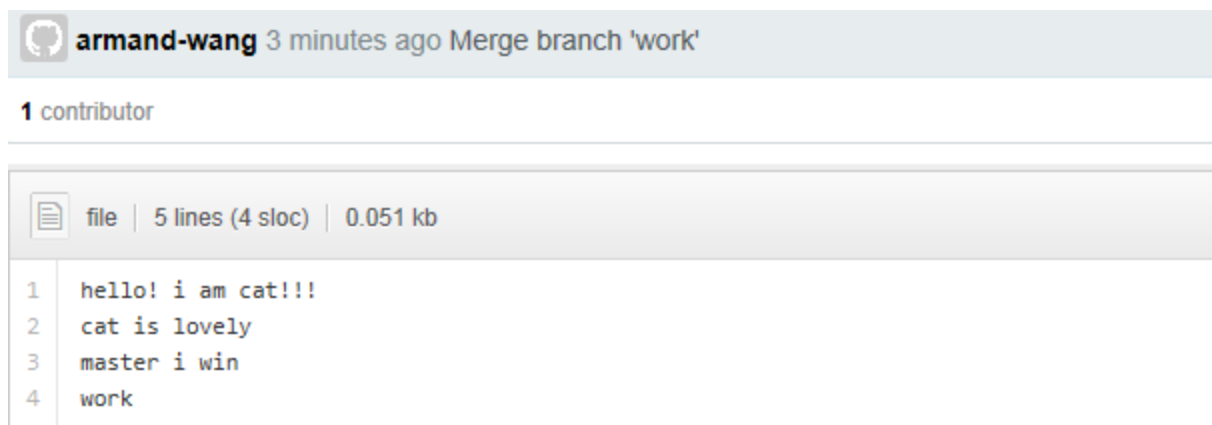
此时切换到主分支下去 merge work 分支看会不会出冲突？

```
$ git merge work
Auto-merging cat.txt
Merge made by the 'recursive' strategy.
 cat.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
B46395@B46395-01 /c/learn git (master)
$
```

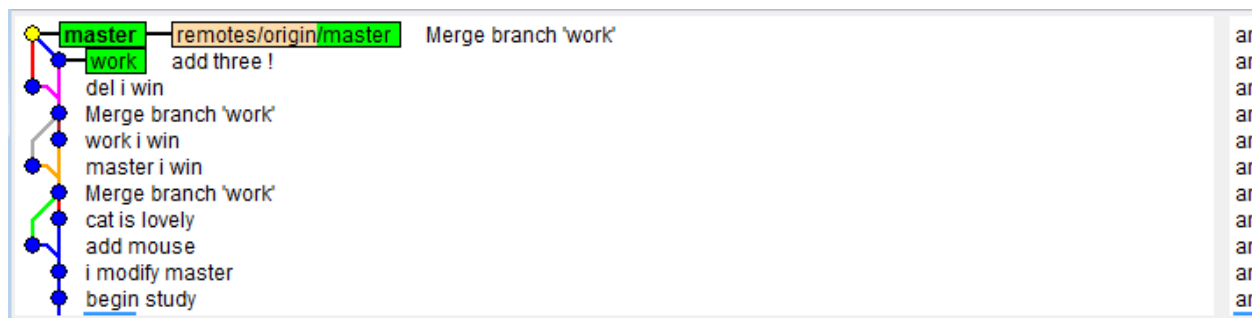
我们看到 git 已经自动将两个不同的 cat.txt merge 完毕，**不能不服啊！！**

## Git: 菜鸟教菜鸟—by Armand

下面我将 push 主分支，push 之后查看 github 上的远端仓库，发现 push 成功：



此时的 code line 已经变成了：



### 九、Git 的基本使用 6:---谁动了我的 master?

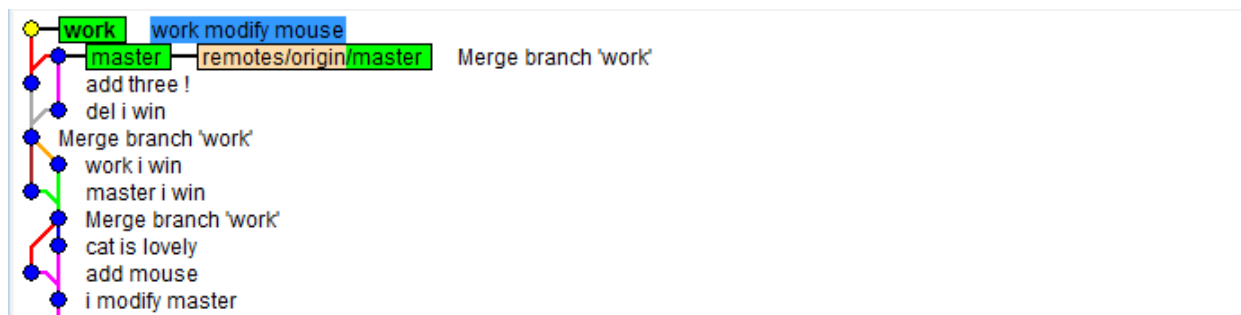
前面的步骤我们一直都是模拟 master 被修改了，然后 pull 下来的过程。但是真实情况下，没有人会自己修改 master。其实我们可以在 github 的仓库里直接修改主分支，然后本地 merge 的时候会出现冲突。

我在 work 分支中添加 mouse.txt 最后一行 ‘work modify this file’ 提交这个修改。

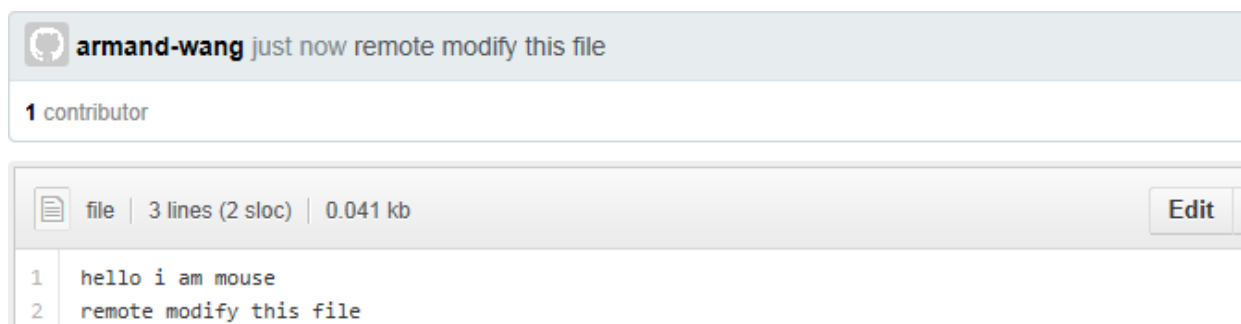
然后在远端仓库，edit mouse.txt，最后一行，‘remote modify this file’ 并保存。

在 work 分支中做了修改之后，code line 如下：

## Git: 菜鸟教菜鸟—by Armand



在远端仓库修改：



这是切换到主分支，pull 下最新的代码

```
$ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From github.com:armand-wang/learngit
 9b823d7..f2c7f74 master -> origin/master
Updating 9b823d7..f2c7f74
Fast-forward
 mouse.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
B46395@B46395-01 /c/learngit (master)
$
```

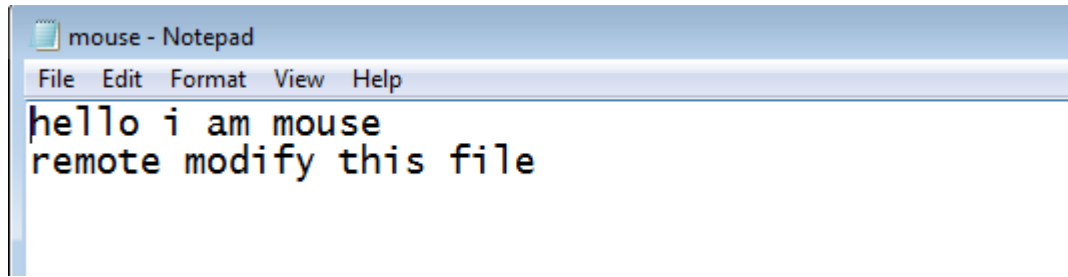
此时我们查看主分支的 git log，第一条显示：

```
Git Bash
commit f2c7f7459df14a50dd9bb50ce41c085ef450cdae
Author: armand-wang <ww_zidongkong@sina.com>
Date: Tue Jul 30 22:47:58 2013 +0800

    remote modify this file
```

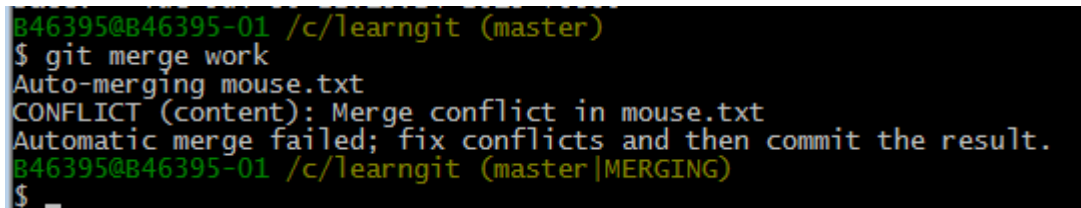
这是我们查看主分支上 **mouse.txt** 的内容：我们发现内容已经是远端修改过的了。





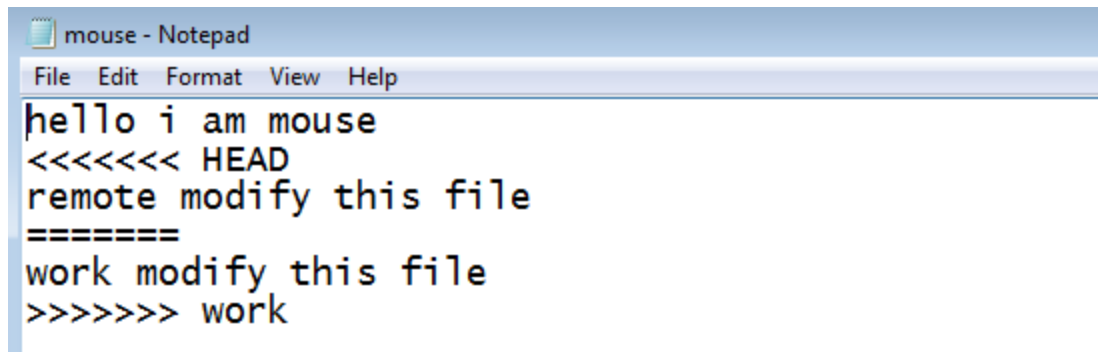
```
mouse - Notepad
File Edit Format View Help
hello i am mouse
remote modify this file
```

此时我们想将工作分支上的修改也 push 上去，此时就会出现冲突（同一文件同一行）



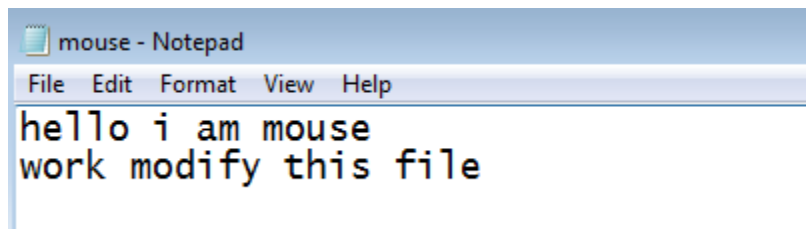
```
B46395@B46395-01 /c/learnGit (master)
$ git merge work
Auto-merging mouse.txt
CONFLICT (content): Merge conflict in mouse.txt
Automatic merge failed; fix conflicts and then commit the result.
B46395@B46395-01 /c/learnGit (master|MERGING)
$
```

根据提示，打开 mouse.txt



```
mouse - Notepad
File Edit Format View Help
hello i am mouse
<<<<<< HEAD
remote modify this file
=====
work modify this file
>>>>>> work
```

我们只保存 remote 的修改（因为我发现别人的修改比我的更好）

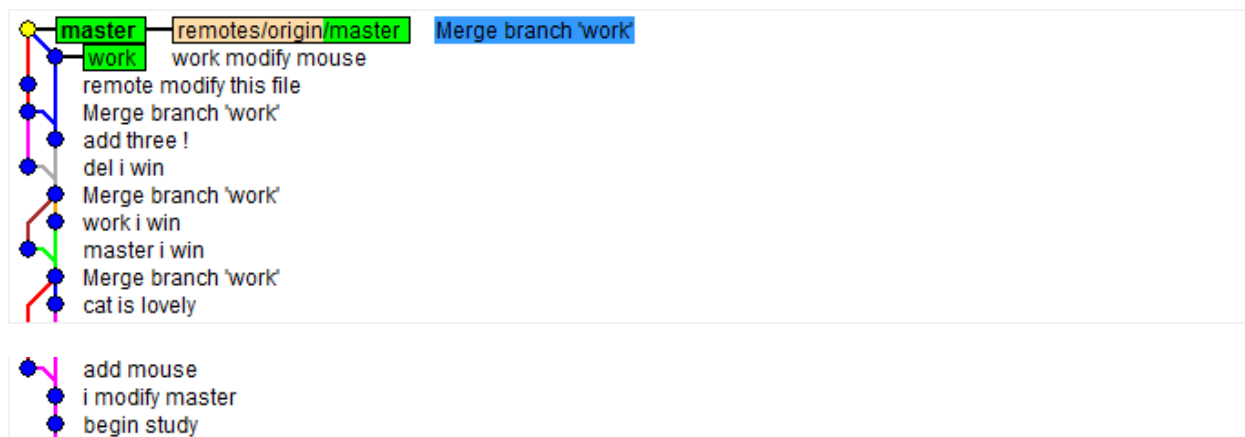


```
mouse - Notepad
File Edit Format View Help
hello i am mouse
work modify this file
```

Git add /git commit 之后我将 push 这次修改

最后的 code line 将变成（拼接图）

## Git: 菜鸟教菜鸟—by Armand



### 十、现在你能分析整个过程吗？

上一个 code line 是不是有点眼花缭乱，但它是我们一步一步搭建的。对比起最初的



它是我们一步一步 commit 起来的，我们的**代码版本一步一个脚印的“成长”**起来。从下面这个 root，你能回忆一下整篇文章的构造过程，分析 code line “成长”的每一步吗？如果可以的话，相信 git 的基本操作你已掌握了。

关于 git，我还将继续学习，本文档持续更新。

**谢谢阅读。**

**By Armand**

**July 30, 2013**