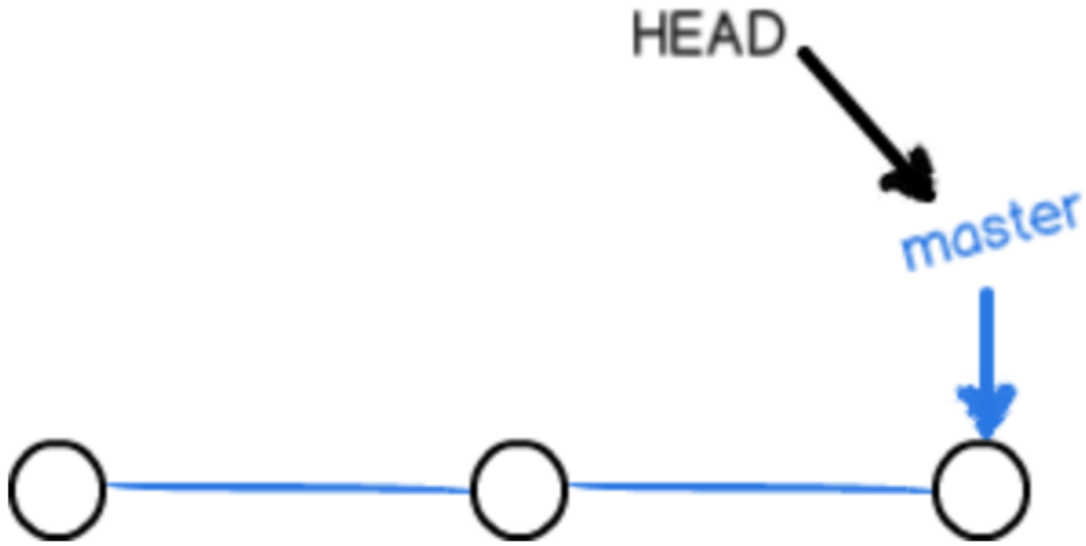
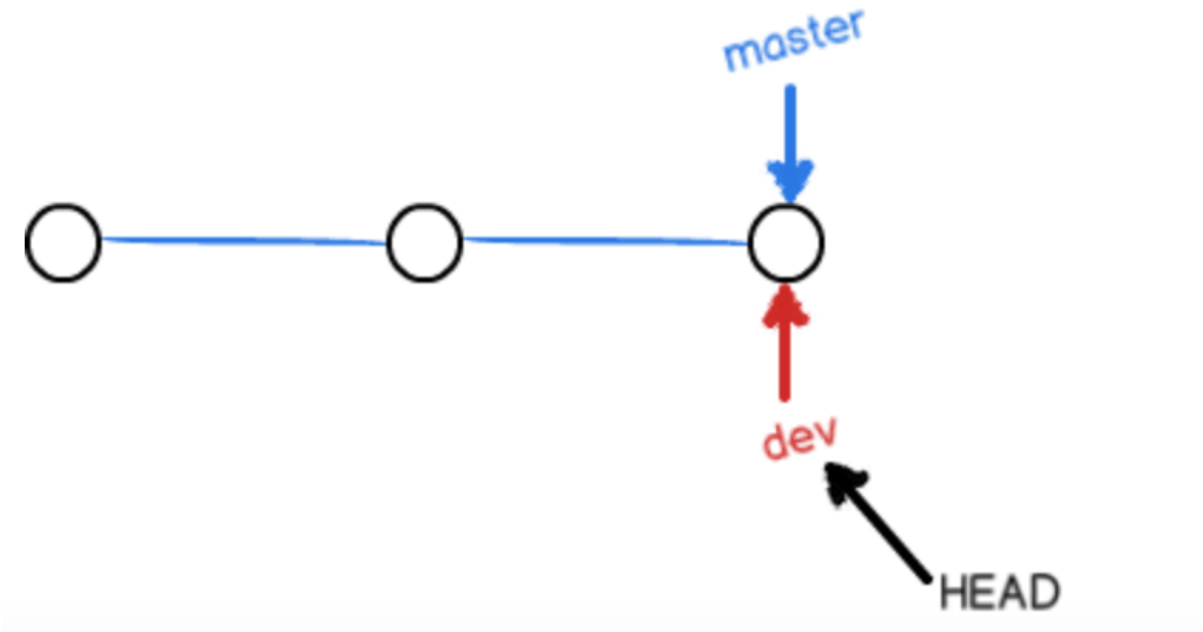
git教程(5) 分支管理是(1)

一 创建于合并分支
1 每次提交，git都会将它们串成一条时间线，这条时间线就是一个分支【主分支！！】。
目前，只有一条时间线，在git里，这个分支叫做 主分支， 即 msater分支。
严格来说，HEAD【指向当前分支！！目前是指向master】不是指向提交【只有master才是指向提交的！！】，而是指向 master 。

2 一开始，master分支就是一条线，git中，master指向最新的提交，再用HEAD指向master。
每一次的提交，master都会向前“延长、加长一步”，如此一来，随着自己的不断提交，master分支的线也越来越长！！

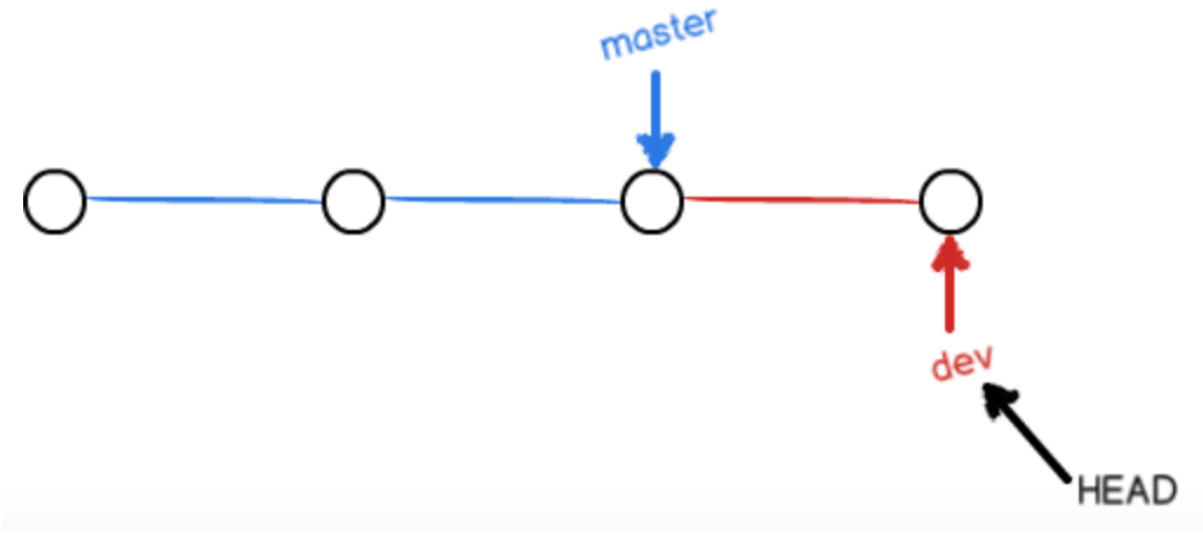


3 创建新分支 dev 之后呢【下图的 HEAD指针指向 表示的是 我们正处于 dev分支上！！】？？
新创建的分支dev “会自动” 指向master相同的提交，再把 HEAD指向dev,表示我们正处于dev分支 上！！！



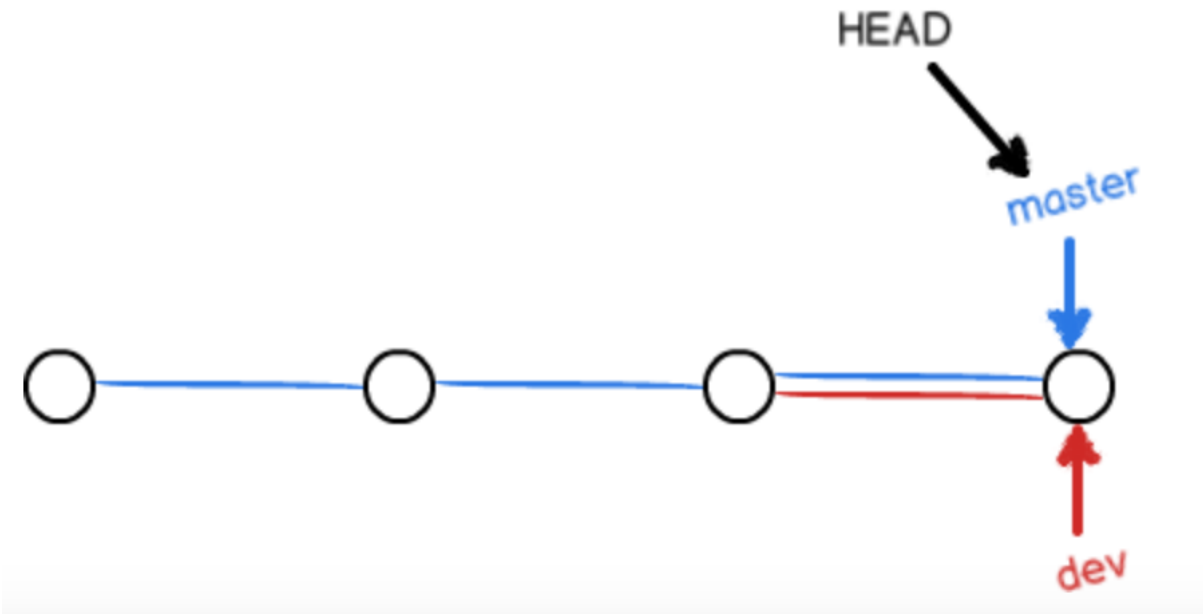
注意：git创建一个分支很快的，因为只是增加了dev指针，改改 HEAD的指向【此时HEAD应指向新创建的分支dev】。工作区的内容没任何的改变！！新分支dev和master分支上的内容一模一样！！

现在对工作区的修改、提交就是针对 dev分支有效了。 ---> 每一次新提交，只有 dev指针向前移动一步，而 master指针位置保持不变！！



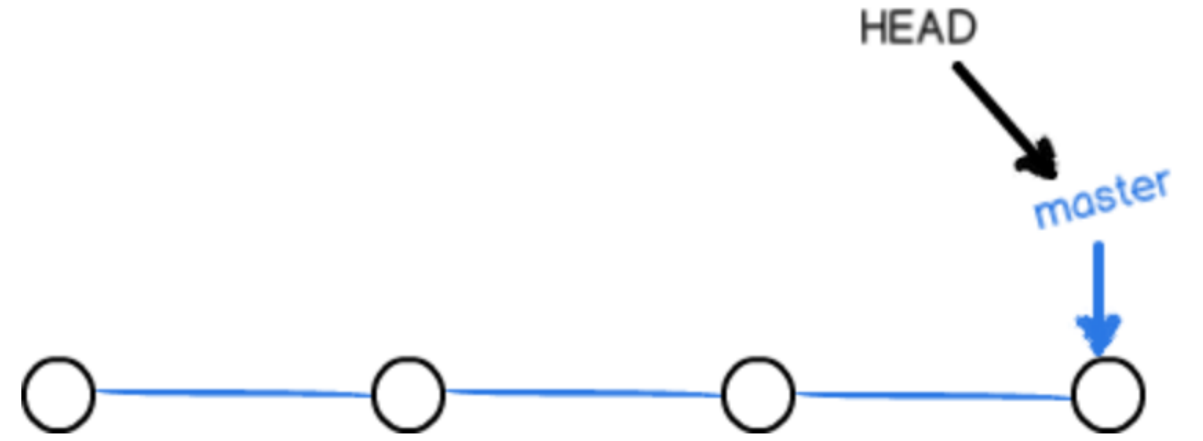
4 开发完dev分支了，如何把 dev 合并到 master分支上呢？？？

最简单的【”不考虑冲突等情况下“】，就是直接 master 指向 dev的当前提交，就完成了合并！！



注意：git的合并分支操作也很快【需注意合并可能带来的冲突问题，内容冲突（“增加”内容、无需手动合并？？） + 编辑冲突（需手动修改？！！）】，也就改改指针的指向，工作区内容保持不变！！

合并完没问题了，可以将 dev分支删除【即把 dev指针删除】。这样一来就好比 master 向前移动了一步！！



5 以上的对应实战

5.1 创建dev分支并切换到 dev分支

```
$ git checkout -b dev
Switched to a new branch 'dev'
```

注意：git checkout -b dev【创建并切换操作，checkout 和 -b能换位置吗？？】 等价于
git branch dev【创建】
git checkout dev【切换】

5.2 git branch 查看当前分支是哪一个！！ 【git branch 会列出所有分支，当前分支就是 前面带* 的 】

然后，用`git branch`命令查看当前分支：

```
$ git branch
* dev
  master
```

5.3 在 dev分支的 readme.txt 文件增一行代码并进行 add、commit 操作。

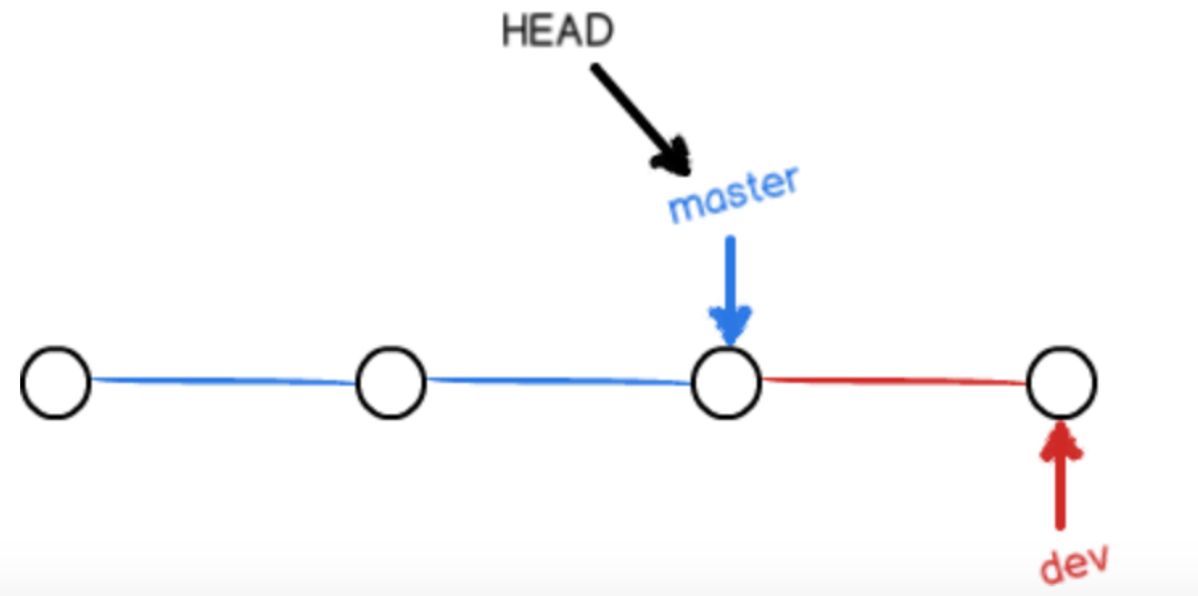
然后，我们就可以在`dev`分支上正常提交，比如对`readme.txt`做个修改，加上一行：

```
Creating a new branch is quick.
```

然后提交：

```
$ git add readme.txt
$ git commit -m "branch test"
[dev b17d20e] branch test
 1 file changed, 1 insertion(+)
```

5.4 dev的开发、add、commit等已完成，此时我们切换到 master分支 【git checkout master】
注意：切回 master 会发现我们在 dev的readme.txt 所做的修改不见了，因为 目前 master、dev 2个分支上的工作区【“暂存区也是？？！”】已经不一样了



5.5 现在我们已经在master分支上了，需要将 dev的修改结果 合并到当前master上进行一个内容的同步！！ 【一般我们是在 dev分支上执行 git merge master，然后再 git push到远程的 dev，最后远程的 dev分支向master发起合并操作！！】

```
$ git merge dev
Updating d46f35e..b17d20e
Fast-forward
 readme.txt | 1 +
 1 file changed, 1 insertion(+)
```

5.6 Fast-forward告诉我们当前合并使用的是 “快进模式”，就是直接把 master指向 dev的当前提交，所以合并速度很快！！！ 这下我们可以安全删除 dev分支了【git branch -d dev。而 git branch -D dev 是强制删除？！！】

注意：因为创建、合并和删除分支非常快，所以 git鼓励 你使用某分支进行开发，合并后再删除分支，这个直接在 master分支上工作是一样的、但这样更安全！！

6 checkout的近义词 —— switch
6.1 我们可以发现 切换分支git checkout <branch> 与之前 撤销修改git checkout -- <file> 是同一个命令，但是后者多加了 --参数，有点令人困惑。
所以，新版本中的 git提供了 git switch 命令，更加合理、语义话。

6.2 git switch -c dev【创建并切换到dev分支，对比 git checkout -b dev】。
单纯切换分支 git switch dev ,对比 git checkout dev。

7 小结
git鼓励大量使用分支

7.1 查看当前分支指在哪： git branch
创建分支：git branch <branch_name>
切换：git checkout <branch_name> || git switch <branch_name>
创建并切换： git checkout -b <brnach_name> || git switch -c <branch_name>
合并某分支到当前分支【前面 带* 号的】： git merge <branch_name>
删除分支：git branch -d <branch_name> 【注意 -D的“强制删除”？？！ 是的，好像处于活跃状态的 分支也能通过 -D 去删除！！】

二 解决冲突
1 解决冲突步骤
1.1 创建并切换到 新建的 feature1 分支上

```
$ git checkout -b feature1
Switched to a new branch 'feature1'
```

1.2 修改 readme.txt 文件最后一行【注意是修改，所以会产生 “编辑冲突，而不是 内容冲突”，此时 git是无法自动合并编辑带来的冲突的！！】

修改`readme.txt`最后一行，改为：

```
Creating a new branch is quick AND simple.
```

1.3 在 feature1 分支上进行提交：

```
$ git add readme.txt

$ git commit -m "AND simple"
[feature1 14096d0] AND simple
 1 file changed, 1 insertion(+), 1 deletion(-)
```

1.4 切换到 master 分支上准备合并

切换到`master`分支：

```
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
```

git 还会自动提示我们当前 master分支 比远程master分支 要超前一个分支【？？？】

制造“编辑冲突”，在 master分支上 把 readme.txt 最后一行改成与 feature1 不一样的：

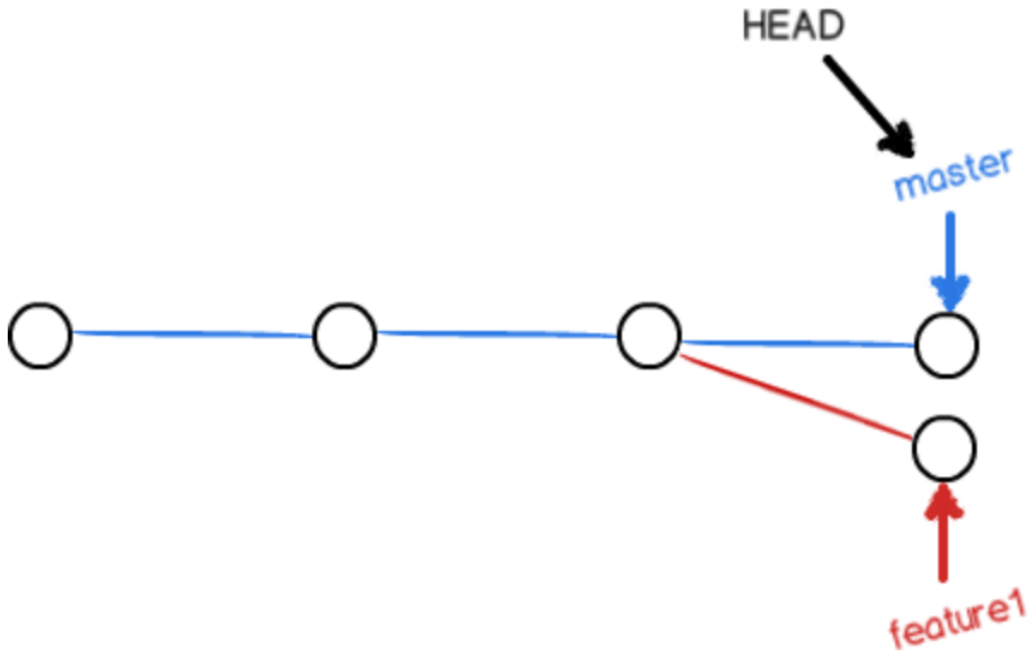
在`master`分支上把`readme.txt`文件的最后一行改为：

```
Creating a new branch is quick & simple.
```

1.5 提交 master所做的修改【注意：master和feature1会产生编辑冲突？！！ ying ga】

```
$ git add readme.txt
$ git commit -m "& simple"
[master 5dc6824] & simple
 1 file changed, 1 insertion(+), 1 deletion(-)
```

现在 master 和 feature1 分支上都有了各自的提交。像这样：



1.6 这种情况下，git 无法快速合并，只能 “试图”【不管内容还是编辑产生的冲突都是 git 先去试图合并？？】 把各自的修改合并起来，但目前的合并会重生冲突。如下：

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

注意：倒数第二行，说的是 产生了 内容冲突、无法自动合并？？？

此时输入 git status

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

1.7 直接在IDE查看readme.txt 的内容：

```
Git is a distributed version control system.
Git is free software distributed under the GPL.
Git has a mutable index called stage.
Git tracks changes of files.
<<<<<<< HEAD
Creating a new branch is quick & simple.
=======
Creating a new branch is quick AND simple.
>>>>>>> feature1
```

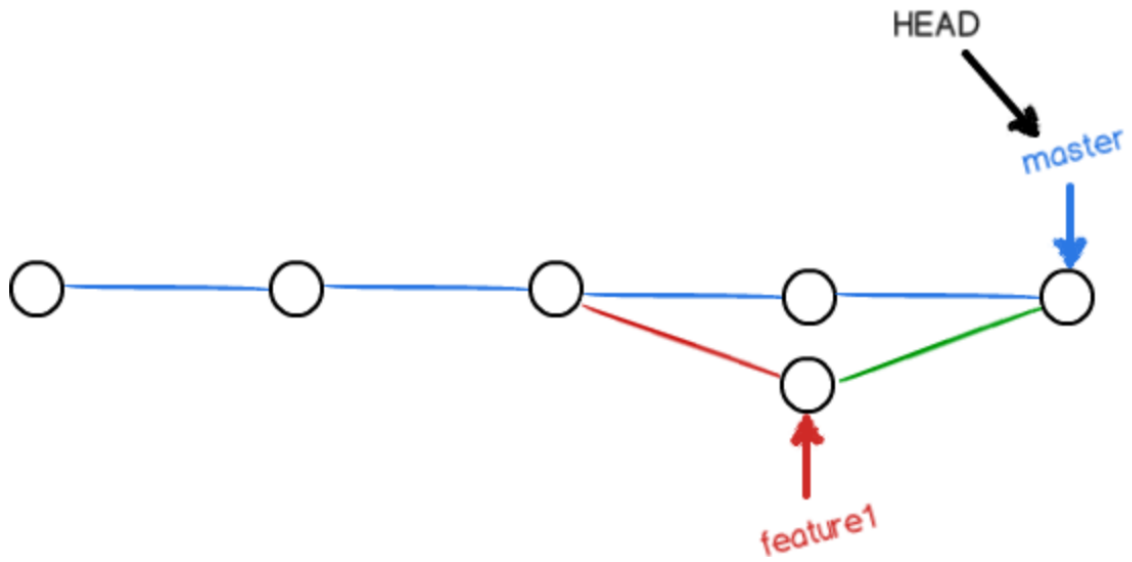
git 用 <<<<<<<< 、 ========== 、>>>>>>>>> 标记不同分支的内容。我们手动合并冲突后，再次提交即可。

```
$ git add readme.txt
$ git commit -m "conflict fixed"
[master cf810e4] conflict fixed
```

注意：需要重新 add 产生冲突的文件，再 comit 操作：

```
$ git add readme.txt
$ git commit -m "conflict fixed"
[master cf810e4] conflict fixed
```

现在各个分支分布情况如下：



git log【按时间倒序排列的！！】 如下：

```
$ git log --graph --pretty=oneline --abbrev-commit
*   cf810e4 (HEAD -> master) conflict fixed
|\
| * 14096d0 (feature1) AND simple
* | 5dc6824 & simple
|/
* b17d20e branch test
* d46f35e (origin/master) remove test.txt
* b84166e add test.txt
* 519219b git tracks changes
* e43a48b understand how stage works
* 1094adb append GPL
* e475afc add distributed
* eaadf4e wrote a readme file
```

2 小结

2.1 当git无法自动合并分支时，就首先解决冲突，解决冲突后，在 add、commit ----> 合并完成。

2.2 解决冲突就是在产生冲突的文件里手动编辑成 我们预期的内容，在提交。

2.3 git log --graph 命令可以看到分支合并图【带上 --pretty参数如 --pretty=oneline 等可以让输出结果更美观！！】

待续