https://www.cnblogs.com/selimsong/p/9059964.html

**[好代码是管出来的——Git的分支工作流与Pull Request](https://www.cnblogs.com/selimsong/p/9059964.html)**

 　　上一篇文章介绍了常用的版本控制工具以及git的基本用法，从基本用法来看git与其它的版本控制工具好像区别不大，都是对代码新增、提交进行管理，可以查看提交历史、代码差异等功能。但实际上git有一个重量级的功能“分支”，git的分支与其它工具的分支不同，git分支的操作完全在本地进行，所以可以快速的创建和切换。

　　版本控制工具除了对代码进行管理外，实际上它还影响了整个软件编码的工作流程，git因为其分支特性使得开发流程发生了变化，本文将从以下几点来介绍分支和git的工作流程：

* [版本控制管理分支简介](https://www.cnblogs.com/selimsong/p/9059964.html#branchIntro)
* [Git的分支](https://www.cnblogs.com/selimsong/p/9059964.html#gitbranch)
  + [分支的基本操作](https://www.cnblogs.com/selimsong/p/9059964.html#branchoperate)
  + [远程分支](https://www.cnblogs.com/selimsong/p/9059964.html#remotebranch)
* [Git基于分支的工作流程](https://www.cnblogs.com/selimsong/p/9059964.html#branchflow)
  + [集中式工作流](https://www.cnblogs.com/selimsong/p/9059964.html#centflow)
  + [功能开发工作流](https://www.cnblogs.com/selimsong/p/9059964.html#featureflow)
  + [Git Flow工作流](https://www.cnblogs.com/selimsong/p/9059964.html#gitflow)
* [Git的分布式工作流](https://www.cnblogs.com/selimsong/p/9059964.html#disflow)
  + [再谈集中式工作流](https://www.cnblogs.com/selimsong/p/9059964.html#centflowagain)
  + [集成管理者工作流](https://www.cnblogs.com/selimsong/p/9059964.html#managerflow)
  + [司令官与副官工作流](https://www.cnblogs.com/selimsong/p/9059964.html#dictatorflow)
* [Pull Request](https://www.cnblogs.com/selimsong/p/9059964.html#pullrequest)
* [Git常用的GUI工具](https://www.cnblogs.com/selimsong/p/9059964.html#GUItool)
* [小结](https://www.cnblogs.com/selimsong/p/9059964.html#summary)

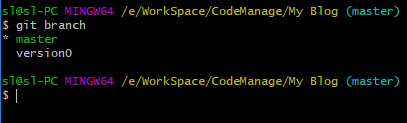
**版本控制管理分支简介**

　　在使用集中式的版本管理工具时，一般会在项目的仓库中创建Trunk(主干)、Branches(分支)、Tag(标记)几个目录，分别用于放置开发代码、代码分支以及代码里程碑，分支的目的是为了开发一些测试性功能或者修复Bug时从开发主线上分开避免互相影响，但是要注意的是集中式的工具创建分支的过程是由服务器完成的，当服务器完成分支的创建后，使用者还需要将分支代码checkout到本地，如果项目较大或者网络较慢，那么checkout将是一个漫长的过程，所以使用集中式工具时分支的创建是相对谨慎的。  
　　对于Git来说分支就是整个仓库的基础(注：当使用git init命令创建一个仓库时，默认会创建一个名为master的分支)，由于Git的本地处理特性，分支的创建基本是一瞬间完成的，正因为这一特性，使用git来进行代码开发的工作流程就有了巨大的变化，如开发功能时创建对应的分支、修复bug(不仅仅是生产bug，任何测试产生的bug都可以创建分支)以及开源项目任何人都可以创建自己的分支进行开发。

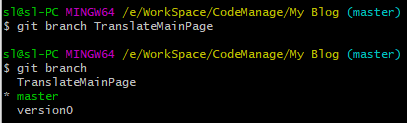
**Git的分支**

**分支的基本操作**

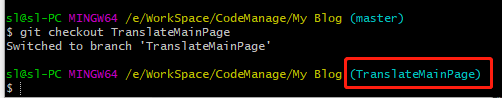
* 查看分支：(git branch)



* 分支的创建：(git branch TranslateMainPage)

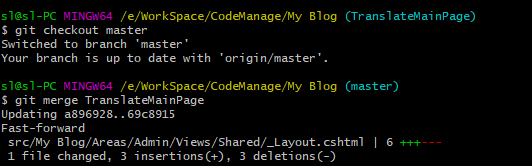


* 分支的切换：(git checkout TranslateMainPage)

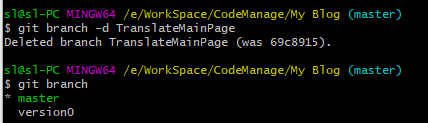


　　注：git checkout -b TranslateMainPage相当于执行了创建和切换两个命令。

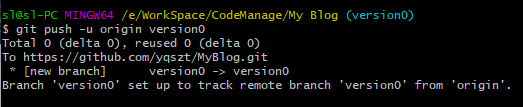
* 分支的合并：(git merge TranslateMainPage)

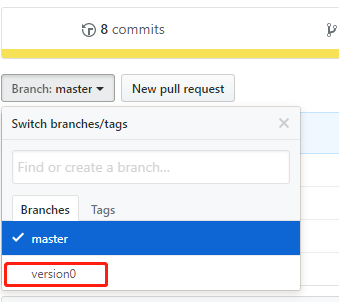


* 分支的删除：(git branch -d TranslateMainPage)



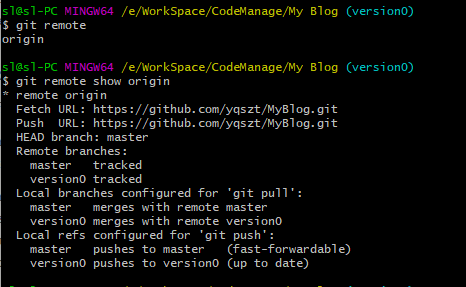
* 将本地分支上传到远程服务器：（git push -u origin version0 　注：-u是--set-upstream的缩写）





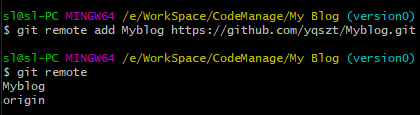
**远程分支以其基本操作**

　　Git的操作都是基于分支的，同时Git作为一个分布式的版本控制工具可以使用远程托管平台来进行代码库托管，那Git的分支是如何在远程平台上体现的呢？  
　　Git中有一个remote命令，它可以用来管理一系列被跟踪或者说被关联的远程仓库（注：remote管理的是仓库），如下图通过git remote以及 git remote show origin来查看远程仓库的信息：

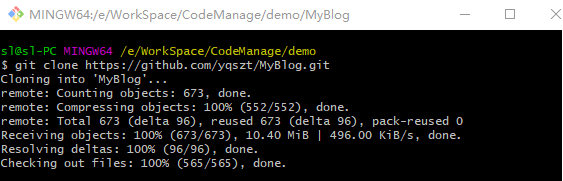


　　这里要注意的是“origin”，它实际上是远程仓库的一个名称，通过容易记忆名称来代替仓库的URL地址更加容易使用，另外如果使用git clone命令来克隆一个远程仓库，那么远程仓库名称会默认为origin。  
　　对于远程分支常用的操作有：

* 添加新的远程仓库：(git remote add Myblog https://github.com/yqszt/Myblog.git，Myblog是本地用来代替后面Url的名称)



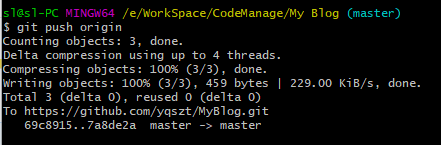
* 克隆一个远程仓库：（git clone https://github.com/yqszt/MyBlog.git）



　　默认创建一个名称为origin的远程仓库：

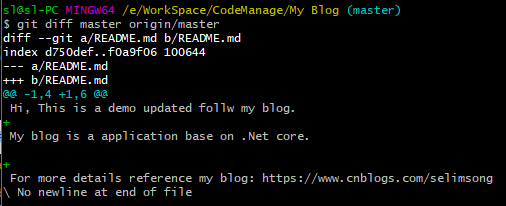
https://images2018.cnblogs.com/blog/640251/201805/640251-20180519124758740-1858452852.png

* 将数据(commit)提交到远程仓库：（git push origin）

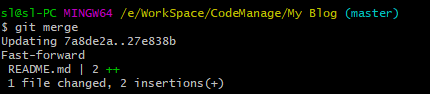


* 从远程仓库拉取更新：（git fetch）

　　注：使用git fetch后，并不会将新的内容更新到工作区域的文件中，所以可以通过git diff master origin/master命令来比较差异：



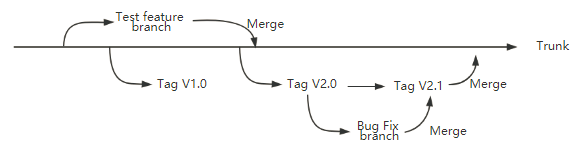
　　同时也可以使用git merge命令来将更新合并到工作区域：



　　注：git pull命令相当于执行了git fetch和git merge两个命令。

**Git基于分支的工作流程**

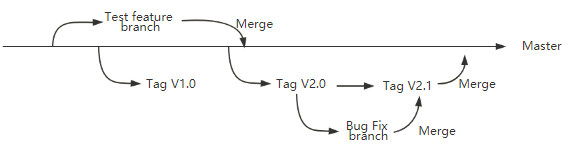
　　之前提到过集中式版本工具中分支的作用是开发一些测试性功能或者修复一些稳定版本的Bug，使用分支可以与开发主线隔离，当完成后再合并到主线中，这种开发流程被称为“集中式工作流”，它的工作流程可看成：



　　1. 以主分支Trunk为核心进行开发，换句话就是开发人员把开发代码都提交到Trunk上，提交之前获取所有代码，并且保证代码能编译成功。  
　　2. 如果有测试性功能，为了与主干代码分离，通过开启分支的形式完成功能开发。(注：这里写测试性功能的原因是，集中式的版本控制工具开启分支代价相对较大，所以在创建分支的时候是谨慎的)。  
　　3. 当开发达到一个里程碑时，通过创建Tag分支来保存里程碑状态，同时Tag出现问题时，可以通过创建Bug修复分支或者直接在Tag分支上修复问题，最终将修复代码合并到Trunk上。  
　　对于分布式的Git来说，由于它创建和切换分支的代价很小，所以可以频繁的创建和切换分支，而分支的功能就是与主干代码隔离，以至于在开发过程中不会因为不完善的功能代码导致主干代码被污染，从而导致无法编译通过，它主要有以下几种开发工作流：

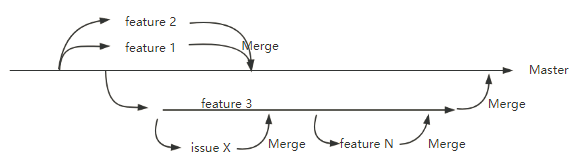
**集中式工作流**

　　集中式工作流就是上面提到集中式版本控制工具中常用的开发流程，以主分支为核心，所有开发人员通过更新主分支代码完成代码的开发工作，同时也会创建一些分支和标签(Git的默认分支是Master)：



**功能开发工作流**

　　功能开发工作流程是以功能为单位进行分支创建，其过程如下：



　　通过创建对应的功能或问题修复分支，完成功能的开发和Bug的修复。这样的好处就是功能与功能之间的代码是隔离的互不影响，利用Git的快速切换分支特性，可以在同一工作目录下同时开发多个功能，且各个功能之间的代码不会互相影响。另外所有新代码均通过合并的方式合并到Master分支，这样代码更容易控制管理。

**Gitflow工作流**

　　Gitflow可以看作是功能开发工作流的完善版本，它除了Master分支、特性分支、Bug修复分支外，还引入了release、develop两个分支来管理发布和开发，而Master只保存稳定版本的代码。



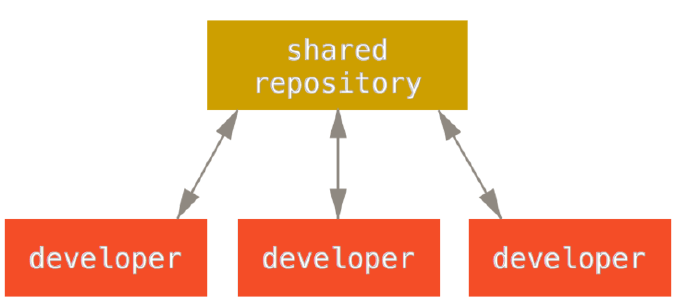
　　（图片来自<https://www.cnblogs.com/cnblogsfans/p/5075073.html>）  
　　总的来说Git就是使用它快速创建和切换分支的特性，在开发过程中通过分支来完成功能的开发、Bug修复以及代码发布。  
　　更多信息可参考：<http://nvie.com/posts/a-successful-git-branching-model/>  
　　　　　　　　　　[https://blog.csdn.net/wwj\_748/article/details/55226044](https://blog.csdn.net/wwj_748/article/details/55226044" \t "_blank)

**Git的分布式工作流**

　　前面介绍了Git的特性之一“分支”的工作流，那么Git的特性之二“分布式”又会对开发模式带来什么样的变化？

**再谈集中式工作流**

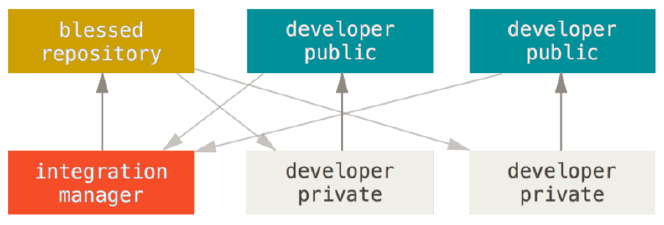
　　为什么又是集中式工作流？文章前面介绍的集中式工作流主要偏重于“分支”，所有工作的内容提交到一个Trunk或者Master的分支上。  
　　而这里的集中式工作流是针对与代码仓库来说的，所有开发人员使用同一个代码仓库进行协同工作，Git中使用集中式工作流时还可以采用特性分支或者Git Flow工作流来体现Git分支带来的便利(注：如果一个项目的贡献者只有一个人的话，实际上集中式工作流联合特性或Git flow来进行开发是最适合的)：



　　在使用集中式版本控制工具时，使用的就是集中式工作流，所有的开发人员共享一个代码仓库，当其中一人提交代码时需要先更新其它人的提交，可能会出现代码冲突需要合并，还有可能会将其它人的提交覆盖掉，同时由于无法保证代码质量，甚至会出现引入了其它开发人员的代码导致编译不通过、测试不通过等等问题，所以在使用集中式工作流程的时候最不能缺少的就是“沟通”。  
　　对于开源项目来说开发人员来自全世界，其沟通成本远远大于本地团队，那么作为开源项目使用最广泛的版本控制工具，它是如何解决协同开发问题？

**集成管理者工作流**

　　Git中可以创建多个仓库，集成管理者工作流的核心就是项目的主仓库由“集成者”负责，其它开发人员拥有自己的仓库，开发者把完成的工作提交到自己的公开库中，然后“集成者”从这些公开库中拉取代码，最终合并到主仓库中，如下图：



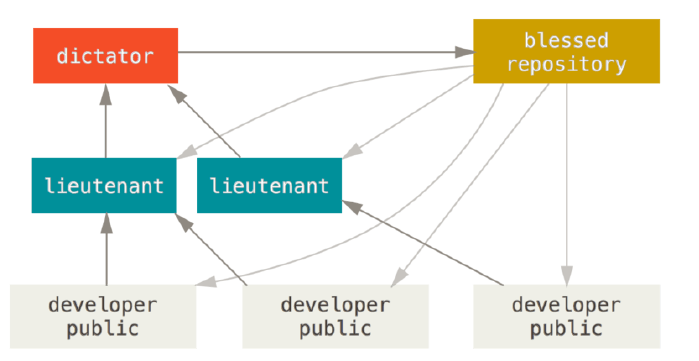
　　这样做有以下几个好处：

* 开发人员有自己的代码库，减少了更新、合并等操作(注：更新、合并的根源在于不同开发任务之间的依赖，如果依赖严重，那么更新、合并是不可避免的，最理想的情况是没有依赖，那么开发人员只需完成自己的工作提交即可)。
* 所有代码合并由“集成者”完成合并，而一般“集成者”由经验丰富的程序员担任，代码合并的过程强制进行了代码复审，对于代码的质量是可控的，有效保证主项目代码的干净整洁。
* 由于代码的复审，开发人员在提交代码时也不会太随意，变相提高了代码质量。

　　但是相对于集中式的工作流来说由于需要等待合并，提交工作也比较复杂，所以开发效率会相对降低。

**司令官与副官工作流**

　　司令官与副官工作流是集成管理者工作流的拓展，引入了多级“集成者”来完成多级的代码合并操作，该模式适用于复杂的多级管理的项目开发：

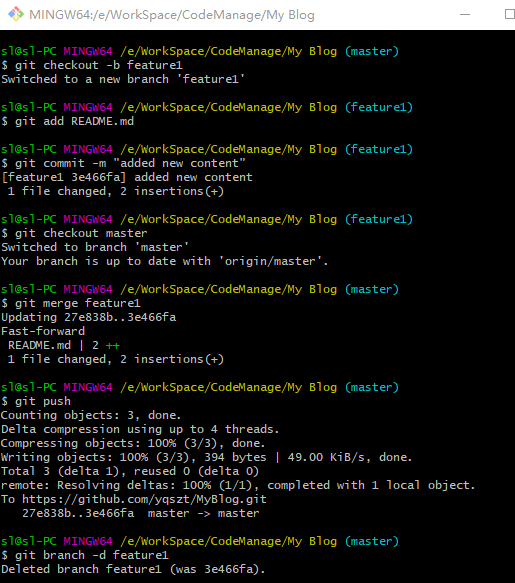


　　更多关于Git分布式工作流的内容可参考：[https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows](https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows" \t "_blank)

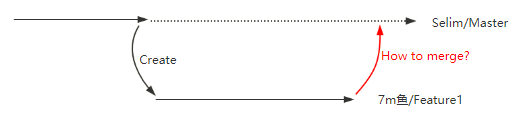
**Pull Request**

　　在Git中无论是集中式工作流还是集成管理者工作流，它都有一个核心的操作就是合并代码，对于集中式工作流来说，当分支完成开发后，需要将代码进行合并，一般是将分支代码合并到远程的如Master或Develop之类的长期分支上，其流程如下：  
　　1. 创建一个功能分支feature1(git checkout -b feature1)。  
　　2. 在分支上完成功能并提交(git add & git commit)。  
　　3. 切换到master分支执行合并操作，并将更新推到远程仓库(git checkout master, git merge feature1, git push)。  
　　4. 删除特性分支(git branch -d feature1)。

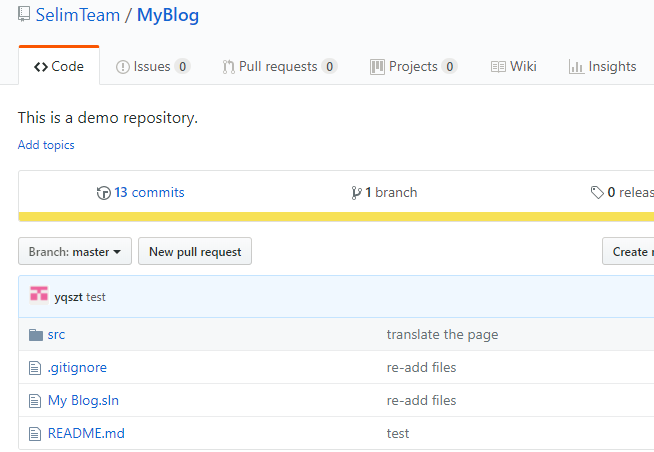
　　过程如下图所示：



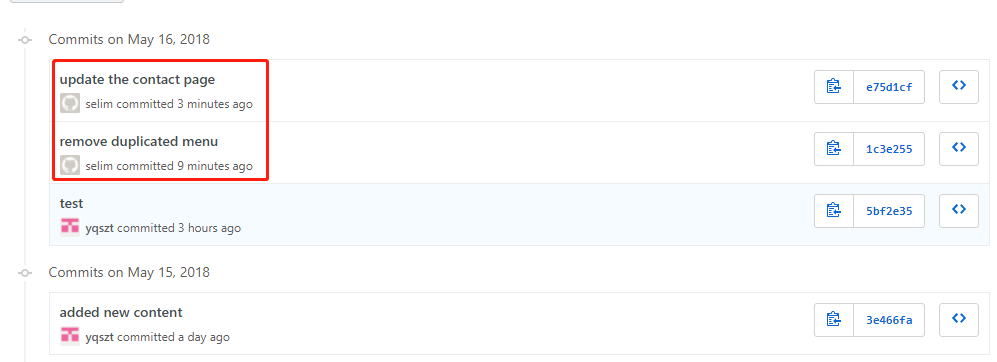
　　但是对于集成管理者工作流来说，集成管理者要如何知道有代码需要合并？要如何合并代码？Git中引入了pull request这一功能彻底的改变了代码的合并方式，这一特性也让其成为开源专用的版本控制工具。  
　　pull request是什么？用中文翻译过来是“拉请求”，假设以下场景：  
　　1. Selim开发了一个应用程序My Blog，并通过某一Git远程托管平台对代码进行了托管。  
　　2. 7m鱼复制了Selim托管的库，然后在App上添加了一个新功能feature1。  
　　3. 现在7m鱼想要将新功能合并到Selim的分支上应该如何操作？如下图所示：



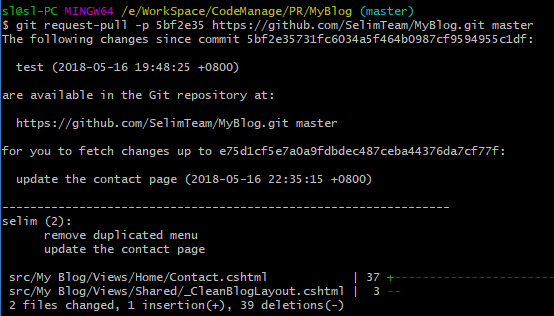
　　首先可以想到的就是使用上面提到的方法切换到Selim的master分支，然后执行git merge Feature1命令，但是如果7m鱼没有Selim/Master的修改权限呢？Selim/Master是属于Selim的，7m鱼无法修改(典型的集成管理者模式，这里“Selim”就是集成管理者)，为了解决这个问题Git实现了“Pull Request(拉请求)”，注意是“拉(pull)”不是“推(push)”，这个请求的目的是让仓库所有者来“拉”取变化，由所有者来决定合并还是拒绝，所有者可以根据功能是否合理、代码是否正确、易读等信息进行判断，这实际上就是CodeRview的过程。  
　　下面创建一个新的代码仓库来演示Git的Pull Request，Pull Request的要求就是需要两个远程分支(仓库)进行合并(代码拥有者的分支和代码贡献者的分支)：  
　　1. 克隆My Blog代码，创建一个新的远程仓库(本例使用GitHub作为托管平台，可以直接fork)：  
　　git clone https://github.com/yqszt/MyBlog.git  
　　git remote add other https://github.com/SelimTeam/MyBlog.git  
　　git push -u other  
　　新建的远程仓库：



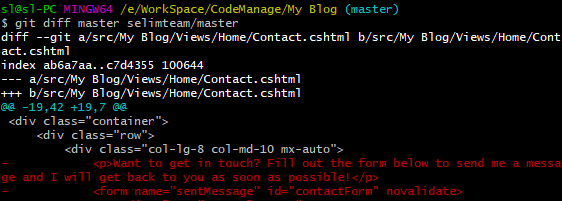
　　2. 在克隆的代码中修改内容并提交：



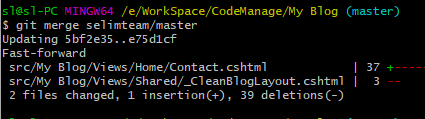
　　3. 要将这两次提交生成“pull request”：  
　　使用git request-pull命令生成拉请求信息：  
　　git request-pull -p 5bf2e35 https://github.com/SelimTeam/MyBlog.git master



**其中p代表输出详细内容(代码的差异)，5bf2e35对应的是提交的hash，代表更新的内容是从哪一个提交开始，url代表的是贡献者的仓库地址，最后的master代表更新内容结束的提交，默认是分支的最新提交**。  
　　4. 将pull request信息告知作者，作者将会知道贡献者的仓库地址、分支、从哪一个提交开始、哪一个提交结束，并且带有详细的变更信息。  
　　注：这里的告知是通过邮件等方式将上面request-pull命令生成的信息发送给作者，github等平台上提供的pull request功能是由平台自己实现的通知方式，关于github上的pull request后续介绍。  
　　5. 作者添加贡献者的远程仓库，获取并将更新合并到主分支：  
　　git remote add selimteam https://github.com/SelimTeam/MyBlog.git  
　　git fetch selimteam master  
　　git diff master selimteam/master



　　git merge selimteam/master

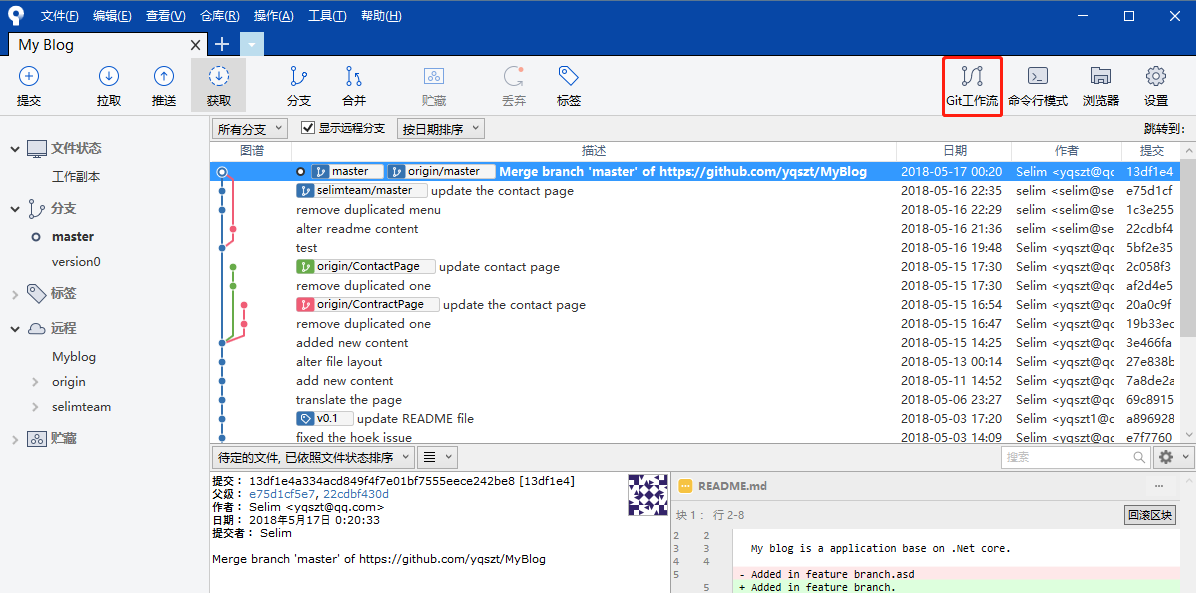


　　git push  
　　以上就完成了一次通过pull request像作者贡献代码的流程。

**Git常用的GUI工具**

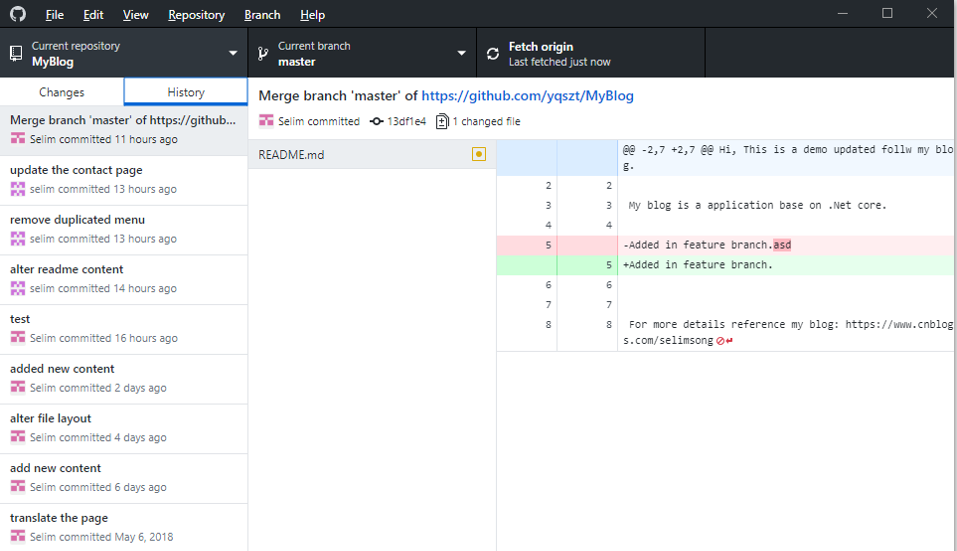
　　从上一篇文章开始都是介绍如何通过命令行的方式使用Git进行代码管理，但在前面的文章中就提到过Git除了原生的命令模式还有GUI模式，GUI主要是针对Git的命令进行封装然后提供了一些更便利的功能来简化使用、提高开发效率。  
　　Git中常用的GUI工具有以下几种：

* SourceTree:一个开源的Git GUI工具，有一个重要的点是它提供了对git flow的支持。

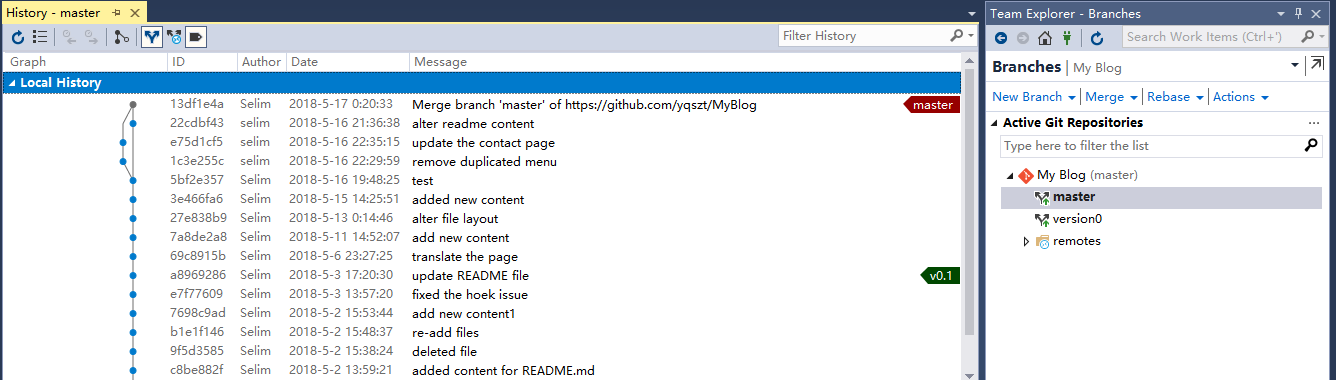


<https://www.sourcetreeapp.com/>  
　　安装参考：[https://www.cnblogs.com/cheese320/p/8876782.html](https://www.cnblogs.com/cheese320/p/8876782.html" \t "_blank)

* GitHub For Desktop:GitHub的GUI客户端，可以通过它直接提交pull request(GitHub的PullRequest)。



* Visual Studio:VS在团队资源管理器中集成了Git的支持，可以在修改完成代码后便捷的进行代码的提交、push等操作。



　　Git的GUI工具有很多，可以通过该链接查找：[https://git-scm.com/download/gui/win](https://git-scm.com/download/gui/win" \t "_blank)

**小结**

　　本文主要介绍了Git分支和Git的工作流，Git的工作流分为两个方面“分支工作流”和“分布式工作流”，两种工作流是混合在一起使用的，前者是用分支对代码进行隔离，后者使用多个远程库以及Pull Request解决了分布式开发、合并的问题。  
　　文章的最后介绍了常用的Git GUI工具，在实际开发中选择适合的GUI工具可以大大的提高开发效率。

参考：  
　　<https://git-scm.com/docs/git-request-pull>  
　　<https://blog.csdn.net/vbirdbest/article/details/51122637>  
　　<https://longair.net/blog/2009/04/16/git-fetch-and-merge/>  
　　<http://nvie.com/posts/a-successful-git-branching-model/>  
　　<https://www.cnblogs.com/cnblogsfans/p/5075073.html>  
　　<https://blog.csdn.net/wwj_748/article/details/55226044>  
　　<https://stackoverflow.com/questions/4037928/can-you-issue-pull-requests-from-the-command-line-on-github>

本文链接：[https://www.cnblogs.com/selimsong/p/9059964.html](https://www.cnblogs.com/selimsong/p/9059964.html" \o "view: 好代码是管出来的——Git的分支工作流与Pull Request" \t "_blank)

[好代码是管出来的——浅谈.Net Core的代码管理方法与落地（更新中...）](https://www.cnblogs.com/selimsong/p/8675028.html)

作者：[7m鱼](http://www.cnblogs.com/selimsong/" \t "_blank)  
出处：<http://www.cnblogs.com/selimsong/>  
本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接，否则保留追究法律责任的权利。

分类: [Git](https://www.cnblogs.com/selimsong/category/1200946.html" \t "_blank),[好代码是管出来的](https://www.cnblogs.com/selimsong/category/1218571.html)

标签: [git](https://www.cnblogs.com/selimsong/tag/git/), [git flow](https://www.cnblogs.com/selimsong/tag/git%20flow/), [git branch](https://www.cnblogs.com/selimsong/tag/git%20branch/)

[**好文要顶**](javascript:void(0);) [**关注我**](javascript:void(0);) [**收藏该文**](javascript:void(0);) **[https://common.cnblogs.com/images/icon_weibo_24.png](javascript:void(0);)** **[https://common.cnblogs.com/images/wechat.png](javascript:void(0);)**

[https://pic.cnblogs.com/face/sample_face.gif](http://home.cnblogs.com/u/selimsong/)

[7m鱼](http://home.cnblogs.com/u/selimsong/)  
[关注 - 2](http://home.cnblogs.com/u/selimsong/followees)  
[粉丝 - 208](http://home.cnblogs.com/u/selimsong/followers)

[+加关注](javascript:void(0);)

3

1

[«](https://www.cnblogs.com/selimsong/p/9051137.html)上一篇：[好代码是管出来的——使用Git来管理源代码](https://www.cnblogs.com/selimsong/p/9051137.html)  
[»](https://www.cnblogs.com/selimsong/p/9135514.html)下一篇：[好代码是管出来的——使用GitHub](https://www.cnblogs.com/selimsong/p/9135514.html)