[**Git 分布式工作流程**](http://wangyan.org/blog/git-work.html)

本文是[《Git 分支管理》](http://wangyan.org/blog/git-branch.html)后续，进一步介绍使用git参与项目开发的各种流程、模式，例如在github上大名鼎鼎的fork+pull模式。

同样，本文是[《Pro Git》](http://progit.org/book/zh/)阅读笔记整理，文中不详细之处，可阅读原著补充。

**一、概述**

**1.1、集中式工作流**

一个存放代码仓库的中心服务器，接受所有开发者提交的代码，而后提交者，必须先下载合并服务器上的数据，解决冲突之后才能推送数据到共享服务器上。

SVN和GIT都可以使用该工作流，优点是简单高效，只需给每个人推送数据的权限，就可以开展工作了。

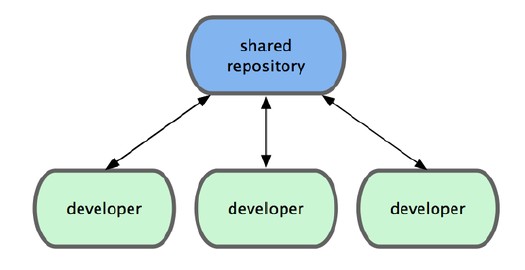
[](http://wangyan.org/pic/g/git-5-1.jpg)

图5-1：集中式工作流

**1.2、集成管理员工作流**

这是 GitHub 上最常见工作流，首先克隆（Fork）某个项目，成为自己的公共仓库，然后按照自己的节奏开发，最后用Pull 工具告知管理员接纳你的贡献。

1、贡献者从官方项目仓库（blessed repository），克隆出各自的私有仓库（developer private）。  
2、贡献者修改代码后，推送数据到自己的公共仓库（developer public）。  
3、贡献者给维护者发送邮件，请求拉取自己的最新修订。  
4. 维护者在自己本地仓库（integration manger）中，将贡献者的仓库加为远程仓库，合并更新并做测试。  
5. 维护者将合并后的更新推送到主仓库（blessed repository）。

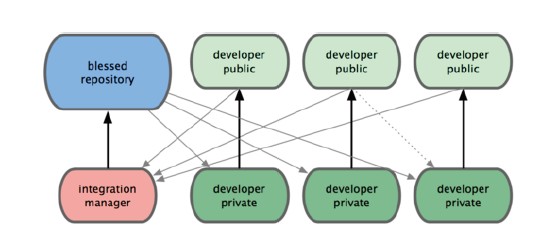
[](http://wangyan.org/pic/g/git-5-2.jpg)

图5-2：集成管理员工作流

**1.3、司令官与副官工作流**

与集成管理员工作流类似，区别是多了一个副官（lieutenant）的角色，他们负责集成项目中的特定部  
分。这种多级别管理方式一般适用于大项目，如linux项目。

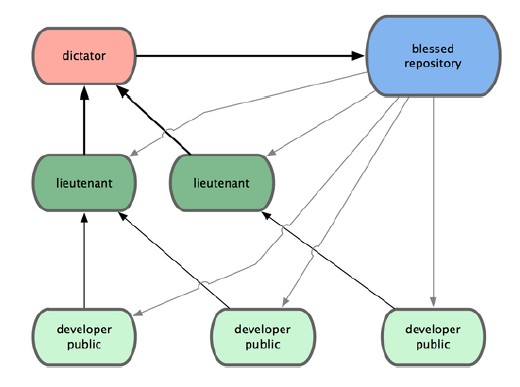
[](http://wangyan.org/pic/g/git-5-3.jpg)

图5-3：司令官与副官工作流

**二、为项目作贡献**

**2.1、提交指南**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | *#请不要在更新中提交多余的白字符（whitespace）*  **git** **diff** --check，    *#请将每次提交限定于完成一次逻辑功能。*  **git**　add --patch    *#请谨记撰写友好的提交说明。*  *#首行简明扼要，另起空行后，再进一步补充说明。*  **git** commit |

**2.2 私有的小型团队**

私有，是指源代码不公开，他人无法访问项目仓库，但你和其他开发者则都具有推送数据到仓库的权限。

私有的小型团队，一般**采用集中式工作流**

1、A和B分别克隆了远程仓库并提交了更新

|  |  |
| --- | --- |
| 1  2 | **git** clone jessica**@**githost:simplegit.git  **git** commit -am 'xxxx' |

2、A推送了项目至远程仓库

|  |  |
| --- | --- |
| 1 | **git** push origin master |

3、B不能直接推送了，首先需要fetch，然后合并到本地，最后再推送。

|  |  |
| --- | --- |
| 1  2  3 | **git** fetch origin　*#首先下载了A推送到服务器的最近更新*  **git** merge origin**/**master *#然后将提交历史合并到本地*  **git** push origin master *#最后B也推送到远程* |

**2.3、私有团队间协作**

成员：John Jessica Josie  
分工：John负责featureA、Josie负责featureB、Jessica参与A和B。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *#Jessica*  $ **git** clone **(**url**)** *#从服务器克隆仓库*  $ **git** checkout -b featureA *#在本地开发A分支*  $ **vim** lib**/**simplegit.rb  $ **git** commit -am 'xxxx' *#图6-1(3309)*  $ **git** push origin featureA *#推送A分支到服务器A分支(Jessica无权推到主分支)* |

Jessica 发邮件给John(在github中使用Pull 工具)，在等待他的反馈之前，Jessica 决定和Josie一起开发featureB

|  |  |
| --- | --- |
| 1  2  3  4  5 | *#Jessica*  $ **git** fetch origin *#更新，包括远程featureB分支。*  $ **git** checkout -b featureB origin**/**master　*#从远程主分支创建一个本地跟踪分支featureB*  $ **vim** lib**/**simplegit.rb  $ **git** commit -am 'add ls-files' *##图6-1(85127)* |

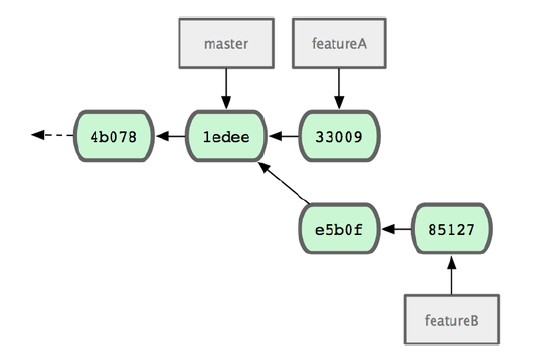
[](http://wangyan.org/pic/g/git-6-1.jpg)

图6-1：Jessica 的更新历史

Jessica正准备将featureB分支推的服务器，但是Josie已抢先推送了featureBee，所以应先更新、合并，然后再推送本地featureB到远程featureBee

|  |  |
| --- | --- |
| 1  2  3  4 | *#Jessica*  $ **git** fetch origin *#图6-2(fba9a)*  $ **git** merge origin**/**featureBee *#图6-2(cd685)*  $ **git** push origin featureB:featureBee *#图6-2(cd685)* |

收到了John的邮件，他对远程featureA分支已做修改，请过目。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | *#Jessica*  $ **git** fetch origin *#下载最新数据*  $ **git** log origin**/**featureA ^featureA *#查看更新了哪些内容*  $ **git** checkout featureA *#切换到本地featureA分支*  $ **git** merge origin**/**featureA *#图6-2(aad88)，把John的修改合并到自己的featureA分支中：*  $ **git** commit -am 'small tweak' *#图6-2(774b3)*  $ **git** push origin featureA *#推送到远程featureA分支* |

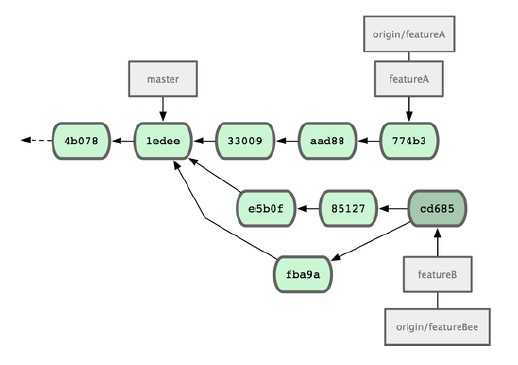
[](http://wangyan.org/pic/g/git-6-2.jpg)

图6-2：Jessica 的更新历史

最后通知集成管理员将featureA 及featureBee分支并入主线。#图6-3(5399e)

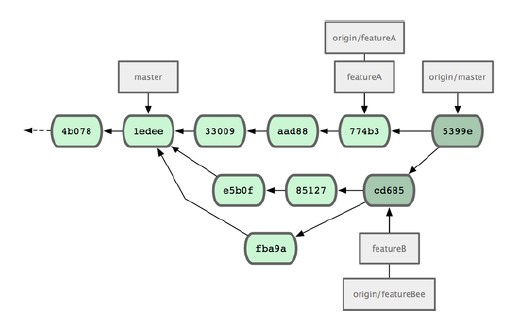
[](http://wangyan.org/pic/g/git-6-3.jpg)

图6-3：管理员合并A、B分支后的历史

**2.4、公开的小型项目**

与私有项目不同，公开项目没有直接更新主仓库分支的权限。

|  |  |
| --- | --- |
| 1  2  3  4  5 | $ **git** clone **(**url**)** *#克隆原始仓库*  $ **cd** project  $ **git** checkout -b featureA *#创建特性分支开展工作*  $ **(**work**)**  $ **git** commit |

首先，点击"Fork"按钮，创建一个自己可写的公共仓库。  
然后，将此仓库添加为本地的第二个远端仓库，命名为myfork  
接着，再把featureA 整个分支推到myfork仓库。  
最后，直接用GitHub提供的pull request工具通知项目管理员。

|  |  |
| --- | --- |
| 1  2 | $ **git** remote add myfork **(**url**)**  $ **git** push myfork featureA |

附：手工发送请求通知

|  |  |
| --- | --- |
| 1  2  3 | *#参数一：是原始分支*  *#参数二：是请求对方来抓取的Git仓库。*  $ **git** request-pull origin**/**master myfork |

建议：随时保持自己的master 分支和官方origin/master 同步，并将自己的工作限制  
在特性分支上，好处是既方便又灵活，采纳和丢弃都轻而易举。

比如现在要开始第二项特性的开发

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $ **git** checkout -b featureB origin**/**master  $ **(**work**)**  $ **git** commit  $ **git** push myfork featureB  $ **(**email maintainer**)**  $ **git** fetch origin |

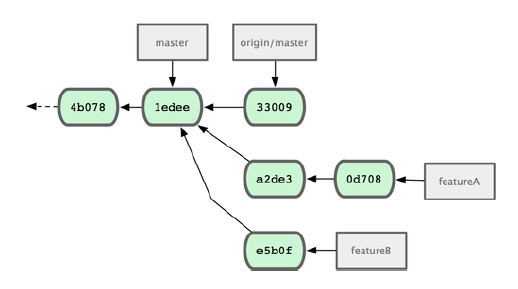
[](http://wangyan.org/pic/g/git-7-1.jpg)

图7-1：featureB 以后的提交历史

假设一：因为代码基准不一致，管理员无法采纳你提交的第一个分支。  
方案一：首先要衍合最新origin/master，解决相关冲突，然后重新提交你的修改

|  |  |
| --- | --- |
| 1  2  3 | $ **git** checkout featureA  $ **git** rebase origin**/**master  $ **git** push -f myfork featureA *#必须使用-f强制推送* |

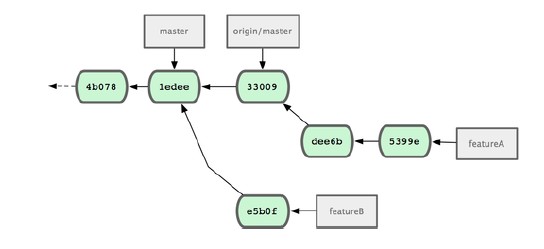
[](http://wangyan.org/pic/g/git-7-2.jpg)

图7-2：featureA 重新衍合后的提交历史

假设二：管理员有意采纳第二个分支后，但还需要做些修改。  
方案二：首先以origin/master为基准，新建特性分支featureBv2，然后合并featureB分支，最后修改提交。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ **git** checkout -b featureBv2 origin**/**master  *# --squash 选项将目标分支上的所有更改全拿来应用到当前分支上*  *# --nocommit 选项告诉Git 此时无需自动生成和记录（合并）提交*  $ **git** merge --no-commit --squash featureB  $ **(**change implementation**)**  $ **git** commit  $ **git** push myfork featureBv2 |

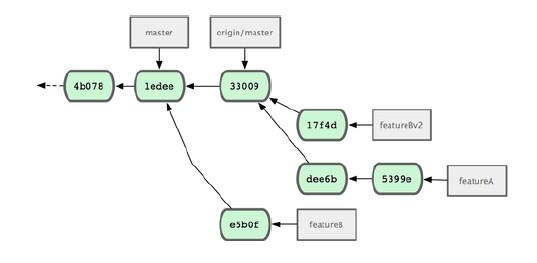
[](http://wangyan.org/pic/g/git-7-3.jpg)

图7-3：featureBv2 之后的提交历史

**2.5、公开的大型项目**

工作流程与上面类似，区别在于提交补丁不需要创建自己可写的公共仓库，只需将每次提交的差异内容以电子邮件的方式依次发送到邮件列表中即可。

|  |  |
| --- | --- |
| 1  2  3 | *# format-patch 命令依次创建补丁文件，并输出文件名*  *# -M 选项允许Git 检查是否有对文件重命名的提交。*  $ **git** format-patch -M origin**/**master |

编辑 ~/.gitconfig 文件，配置imap选项

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **[**imap**]**  folder = "[Gmail]/Drafts"  host = imaps:**//**imap.gmail.com  user = user**@**gmail.com  pass = p4ssw0rd  port = 993  sslverify = **false** |
| 1 | $ **git** send-email **\***.patch　*#发送补丁到邮件列表* | |

......

原文地址 : <http://wangyan.org/blog/git-work.html>