制造工程协同平台

# GIT 学习培训资料

编写人：李瑞东

# Git工作流方式

1、集中式工作流

2、功能分支工作流

3、Gitflow工作流

4、Forking工作流

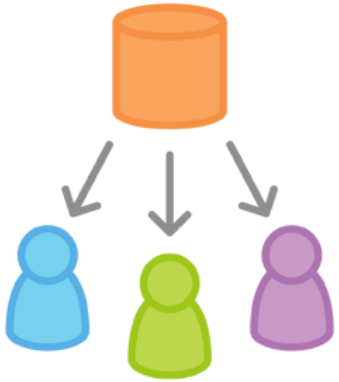
5、Pull Requests

# 一、集中式工作流

**示例：**一个常见的小团队如何用这个工作流来协作的。
有两个开发者小明和小红，看他们是如何开发自己的功能并提交到中央仓库上的。

**1.首先有人创建好一个仓库**

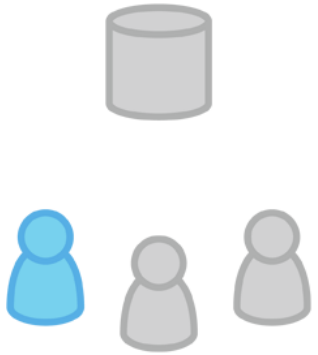
**2.所有人克隆中央仓库**



```
git clone ssh://user@host/path/to/repo.git
```

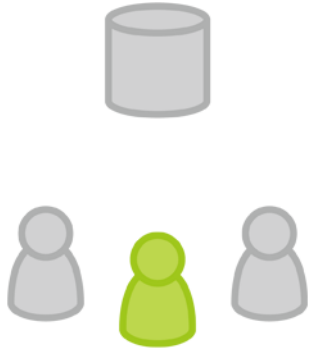
MoWork

**3.小明开发功能**

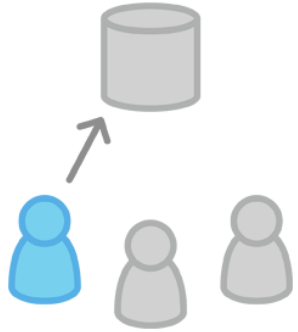


```
git status # 查看本地仓库的修改状态
git add # 暂存文件
git commit # 提交文件
```

**4.小红开发功能**

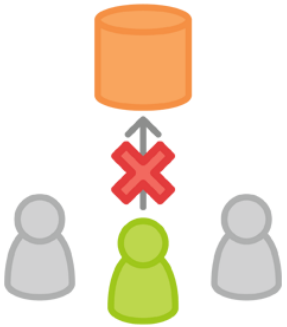


**5.小明发布功能**



```
git push origin master
```

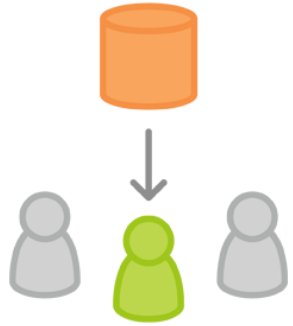
**6.小红试着发布功能（执行和小明一样的操作）**



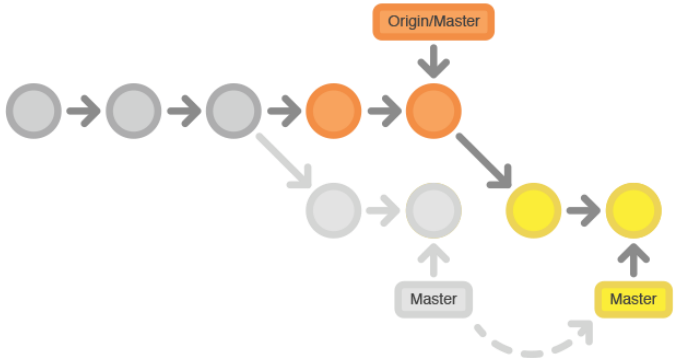
```
error: failed to push some refs to '/path/to/repo.git' hint: Updates were rejected because the tip of your current branch is behind hint: its remote counterpart. Merge the remote changes (e.g. 'git pull') hint: before pushing again. hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

MoWork

**7.小红在小明的提交上rebase**



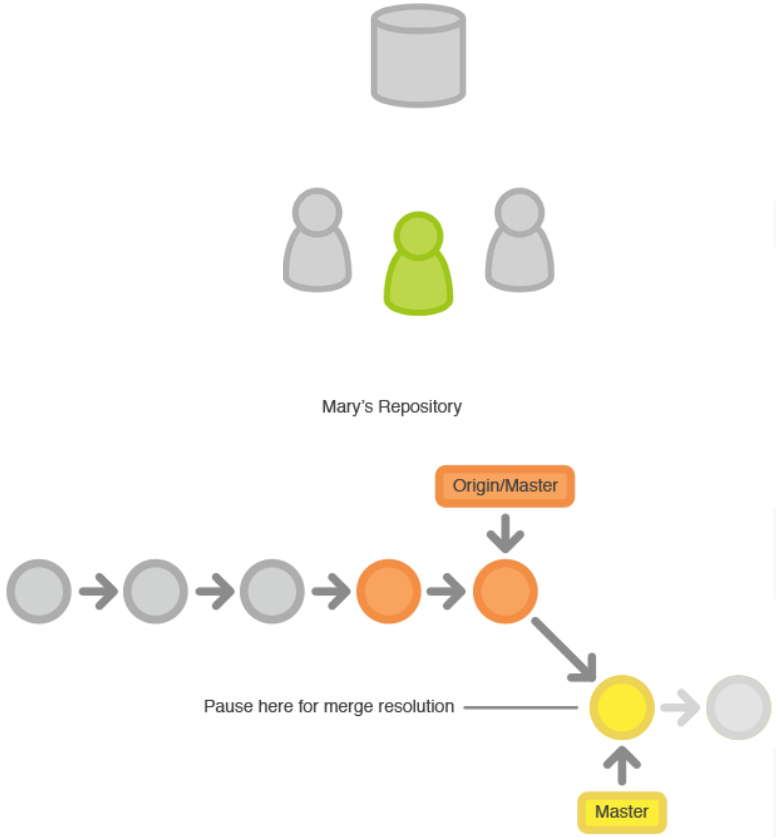
Mary's Repository




git pull --rebase origin master
--rebase选项告诉Git把小红的提交移到同步了中央仓库修改后master分支的顶部

MoWork

## 8.小红解决合并冲突



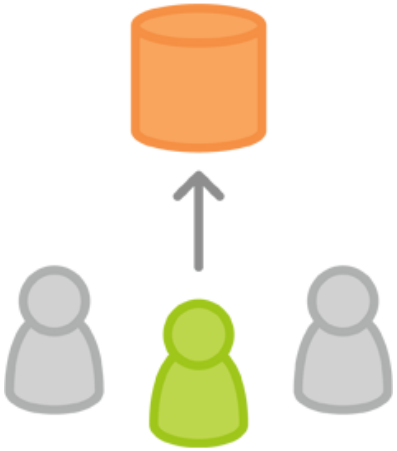

git status 查看问题

git add <some-file>
git rebase --continue

如果问题解决不了，可以执行下面语句返回
git rebase --abort

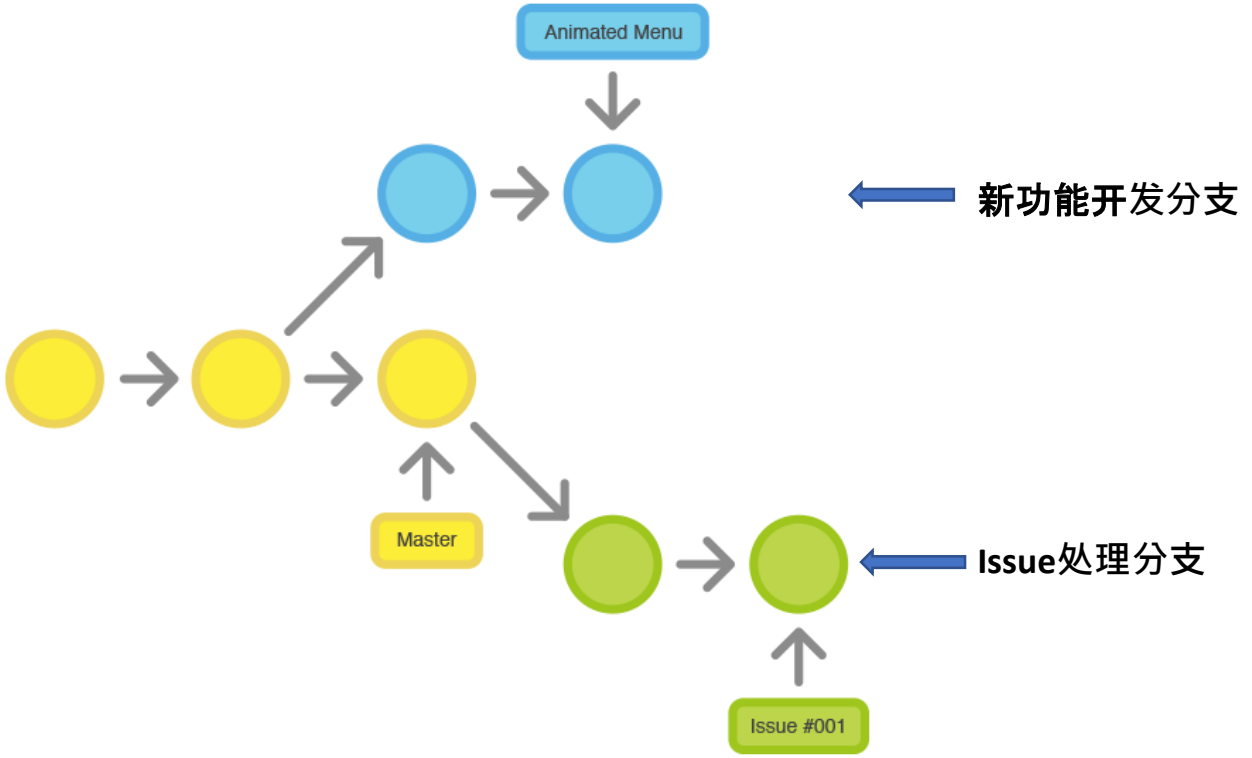
MoWork

**9.小红成功发布功能**



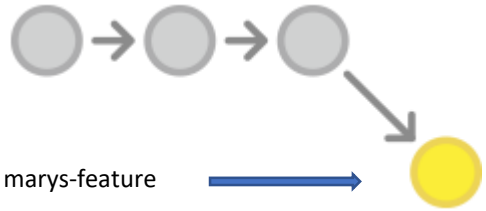
```
git push origin master
```

## 二、功能分支工作流--功能分支工作流以集中式工作流为基础，不同的是为各个新功能分配一个专门的分支来开发。这样可以在把新功能集成到正式项目前，用Pull Requests的方式讨论变更。



# 示例

**1.小红开始开发一个新功能**

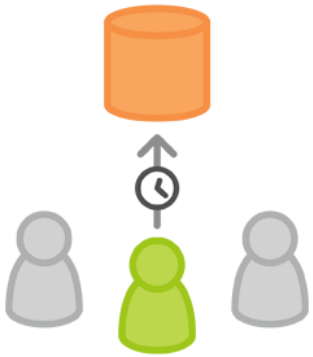


小红创建一个新分支 marys-feature

```
git checkout -b marys-feature master
```

-b选项表示如果分支还不存在则新建分支

**2.小红要去吃过午饭**



```
git status
git add <some-file>
git commit
```

吃饭前

```
git push -u origin marys-feature
```

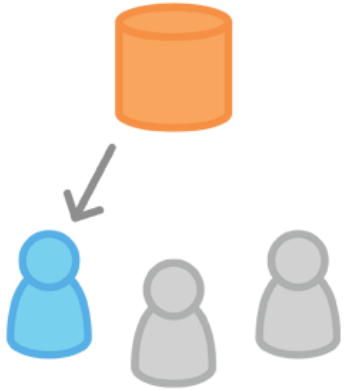
-u选项设置本地分支去跟踪远程对应的分支

MoWork

**3.小红完成功能开发**

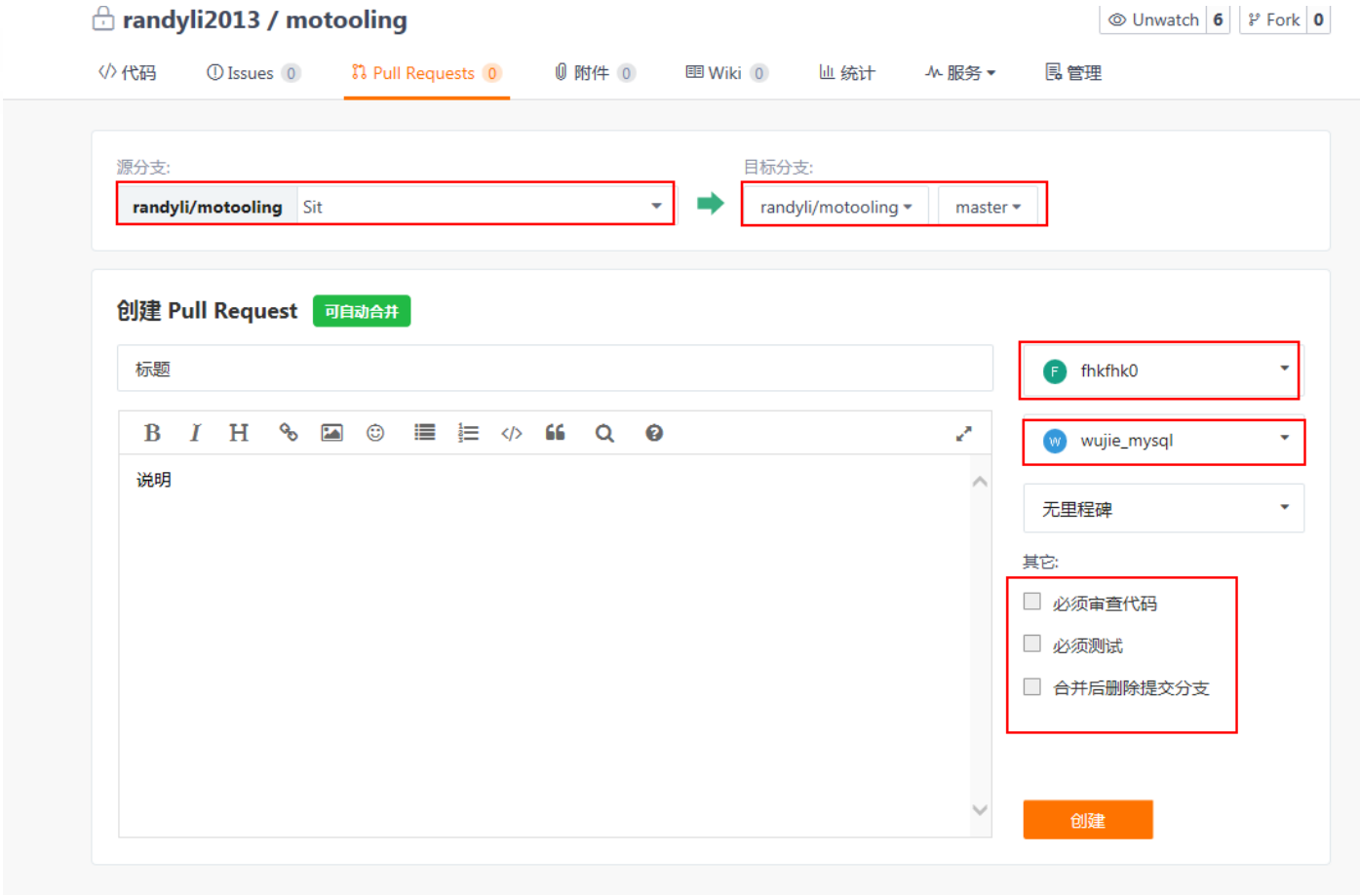
git push
她发起一个Pull Request让团队的其它人知道功能已经完成，请求合并marys-feature到master，团队成员会自动收到通知

**4.小黑收到Pull Request**

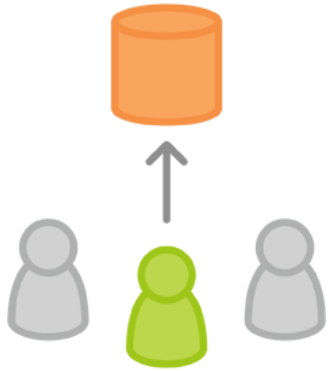


小黑收到了Pull Request后会查看marys-feature的修改。决定在合并到正式项目前是否要做些修改，且通过Pull Request和小红来回地讨论

# 码云上的Pull Requests

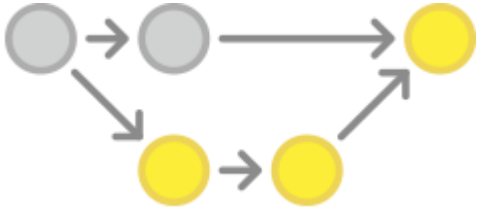


**5.小红再做修改**



如果小黑有需要，也可以把marys-feature分支拉到本地，自己来修改，他加的提交也会一样显示在Pull Request上。

**6.小红发布她的功能**



```
git checkout master
git pull
git pull origin marys-feature
git push
```

无论谁来做合并，首先要检出master分支并确认是它是最新的。然后执行git pull origin marys-feature合并marys-feature分支到和已经和远程一致的本地master分支。 你可以使用简单git merge marys-feature命令，但前面的命令可以保证总是最新的新功能分支。 最后更新的master分支要重新push回到origin

# 三、Gitflow工作流--通过为功能开发、发布准备和维护分配独立的分支，让发布迭代过程更流畅。

MoWork

**1.历史分支**


**2.功能分支**


每个新功能位于一个自己的分支，这样可以push到中央仓库以备份和协作。 但功能分支不是从master分支上拉出新分支，而是使用develop分支作为父分支。当新功能完成时，合并回develop分支。 新功能提交应该从不直接与master分支交互

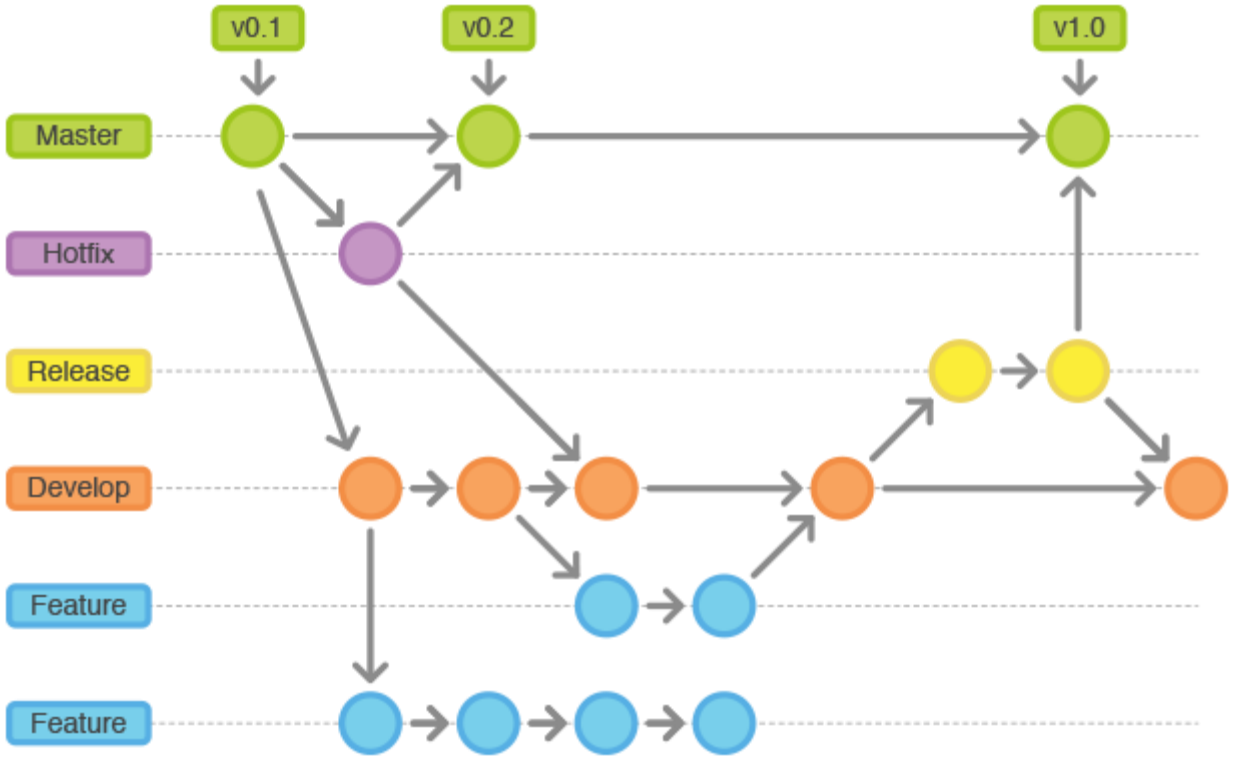
MoWork

**3.发布分支**


用于新建发布分支的分支: develop
用于合并的分支: master
分支命名: release-* 或 release/*

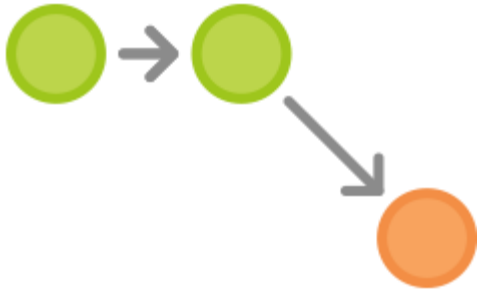
MoWork

**4.维护分支**



# 示例

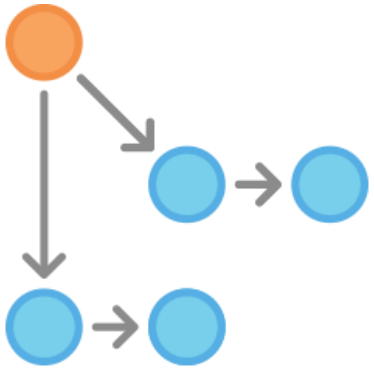
**1.创建开发分支**



```
git branch develop
git push -u origin develop

其他开发者克隆中央仓库
git clone ssh://user@host/path/to/repo.git
git checkout -b develop origin/develop
```

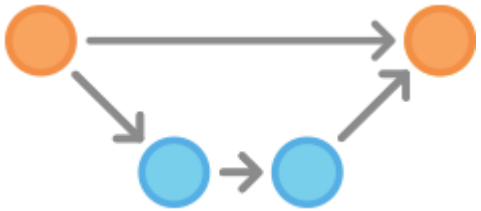
**2.小红和小明开始开发新功能**



```
git checkout -b some-feature1 develop
<-----下面是基本操作----->
git status
git add <some-file>
git commit
```

```
git checkout -b some-feature2 develop
```

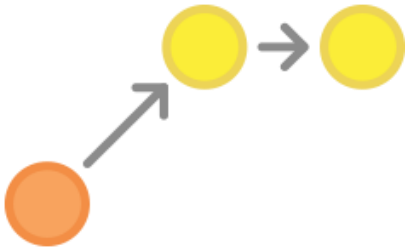
**3.小红完成功能开发**



```
git pull origin develop
git checkout develop
git merge some-feature1
git push git branch -d some-feature1
```

注意：不要合并到master上

**4.小红准备开始发布**

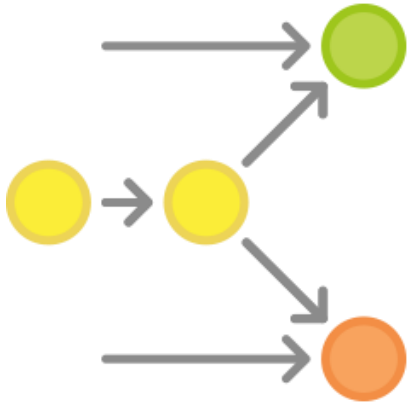


```
git checkout -b release-0.1 develop
```

只要小红创建这个分支并push到中央仓库，这个发布就是功能冻结的。任何不在develop分支中的新功能都推到下个发布循环中。

MoWork

**5.小红完成发布**



```
git checkout master
git merge release-0.1
git push
git checkout develop
git merge release-0.1
git push
git branch -d release-0.1
```

发布分支是作为功能开发（develop分支）和对外发布（master分支）间的缓冲。只要有合并到master分支，就应该打好Tag以方便跟踪

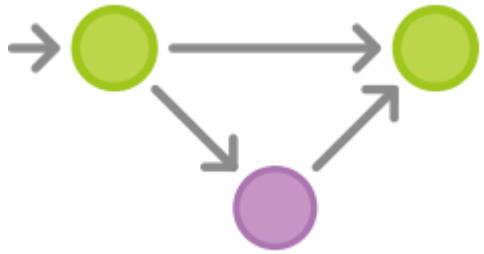
```
git tag -a 0.1 -m "Initial public release" master
git push –tags
```

Git有提供各种勾子（hook），即仓库有事件发生时触发执行的脚本。 可以配置一个勾子，在你push中央仓库的master分支时，自动构建好版本，并对外发布

**6.最终用户发现bug**

某天用户发现一个bug，为了处理bug，小红或小明从master分支拉取一个维护分支，解决问题后合并回master分支上

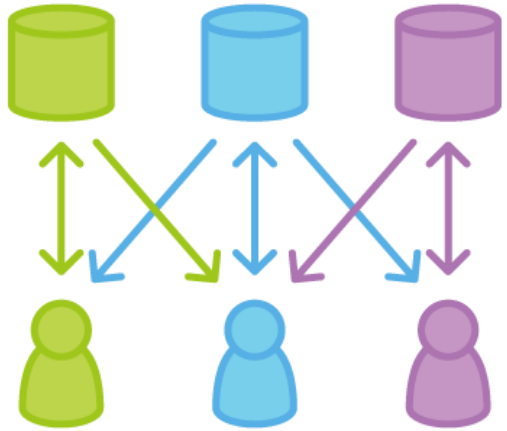
```
git checkout -b issue-#001 master
# Fix the bug
git checkout master
git merge issue-#001
git push
```

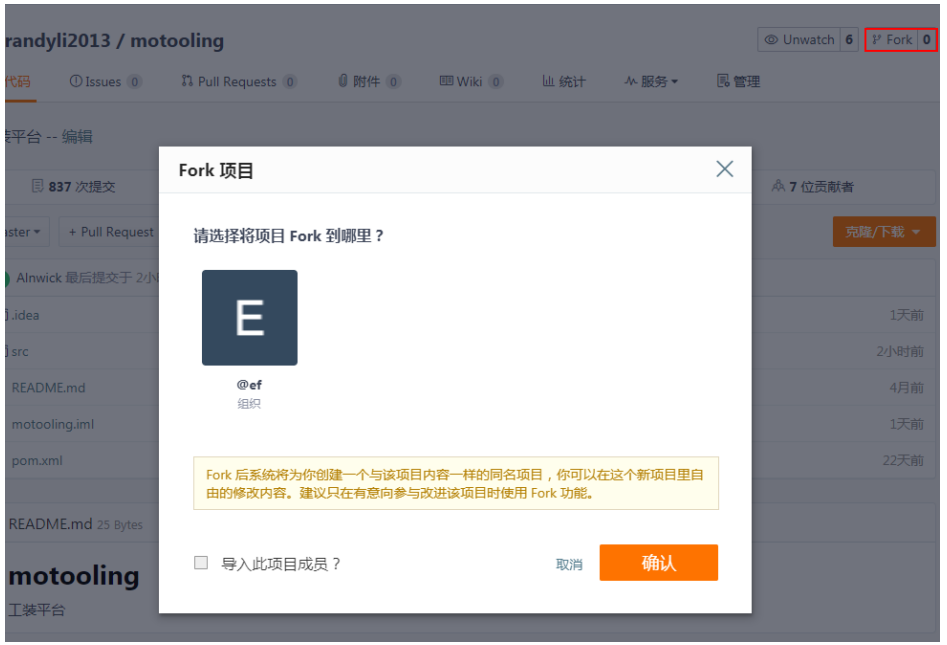


就像发布分支，维护分支中新加这些重要修改需要包含到develop分支中，所以小红要执行一个合并操作。然后就可以安全地删除这个分支了

```
git checkout develop
git merge issue-#001
git push
git branch -d issue-#001
```

# 四、Forking工作流--工作流是分布式工作流，充分利用了Git在分支和克隆上的优势。可以安全可靠地管理大团队的开发者（developer），并能接受不信任贡献者（contributor）的提交，和前面讨论的几种工作流有根本的不同，这种工作流不是使用单个服务端仓库作为『中央』代码基线，而让各个开发者都有一个服务端仓库。





# 示例

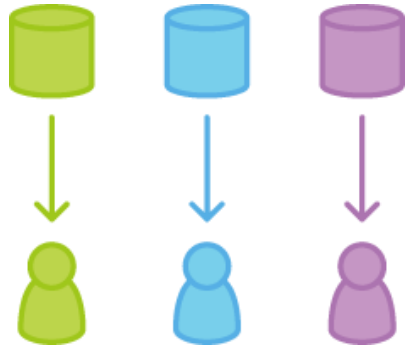
**1.项目维护者初始化正式仓库**

```
ssh user@host
git init --bare /path/to/repo.git
```

**2.开发者fork正式仓库**

MoWork

**3.开发者克隆自己fork出来的仓库**

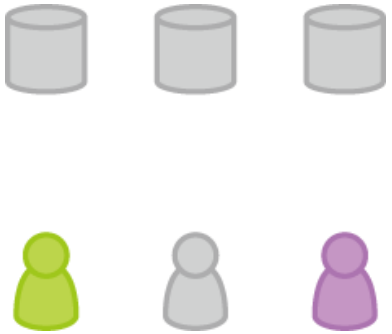


git clone https://user@bitbucket.org/user/repo.git

**相比前面介**绍的工作流只用了一个origin远程别名指向中央仓库，Forking**工作流需要2个**远程别名 —— **一个指向正式**仓库，另一个指**向开**发者自己的服务端仓库。别名的名字可以任意命名，常见的约**定是使用**origin**作**为远程克隆的仓库的别名（这个别名会在运行git clone**自**动创建），upstream（**上游**）**作**为正式仓库的别名。

**git remote add upstream https://bitbucket.org/maintainer/repo**

**4.开发者开发自己的功能**

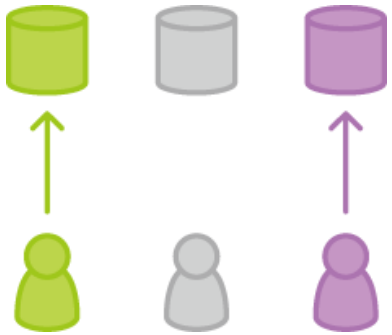


git checkout -b some-feature
# Edit some code
git commit -a -m "Add first draft of some feature“

**所有的修改都是私有的直到push到自己公开**仓库中。**如果正式**项**目已**经往前走了，**可以用**git pull**命令**获得新的提交：
git pull upstream master

MoWork

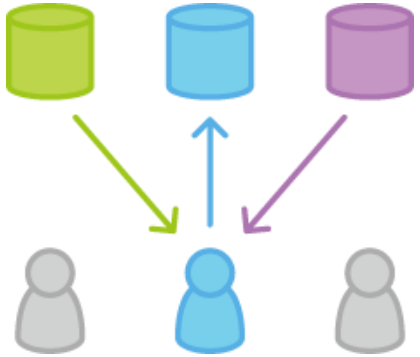
**5.开发者发布自己的功能**



git push origin feature-branch
这里开发者push代码到自己的公开仓库

开发者要通知维护者，想要合并到正式库中，通过Pull Request功能通知维护者

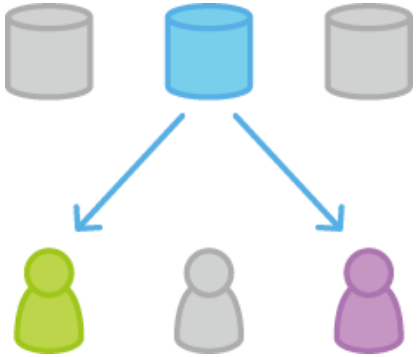
**6.项目维护者集成开发者的功能**



第一种方式：维护者直接在pull request中查看代码
第二年方式：维护者pull代码到他自己的本地仓库，再手动合并
合并中出现冲突的解决方式：
git fetch https://bitbucket.org/user/repo feature-branch
# 查看变更
git checkout master
git merge FETCH_HEAD
问题解决后，需要维护者将本地master push到服务器上，这样其他开发者就能访问到。
git push origin master

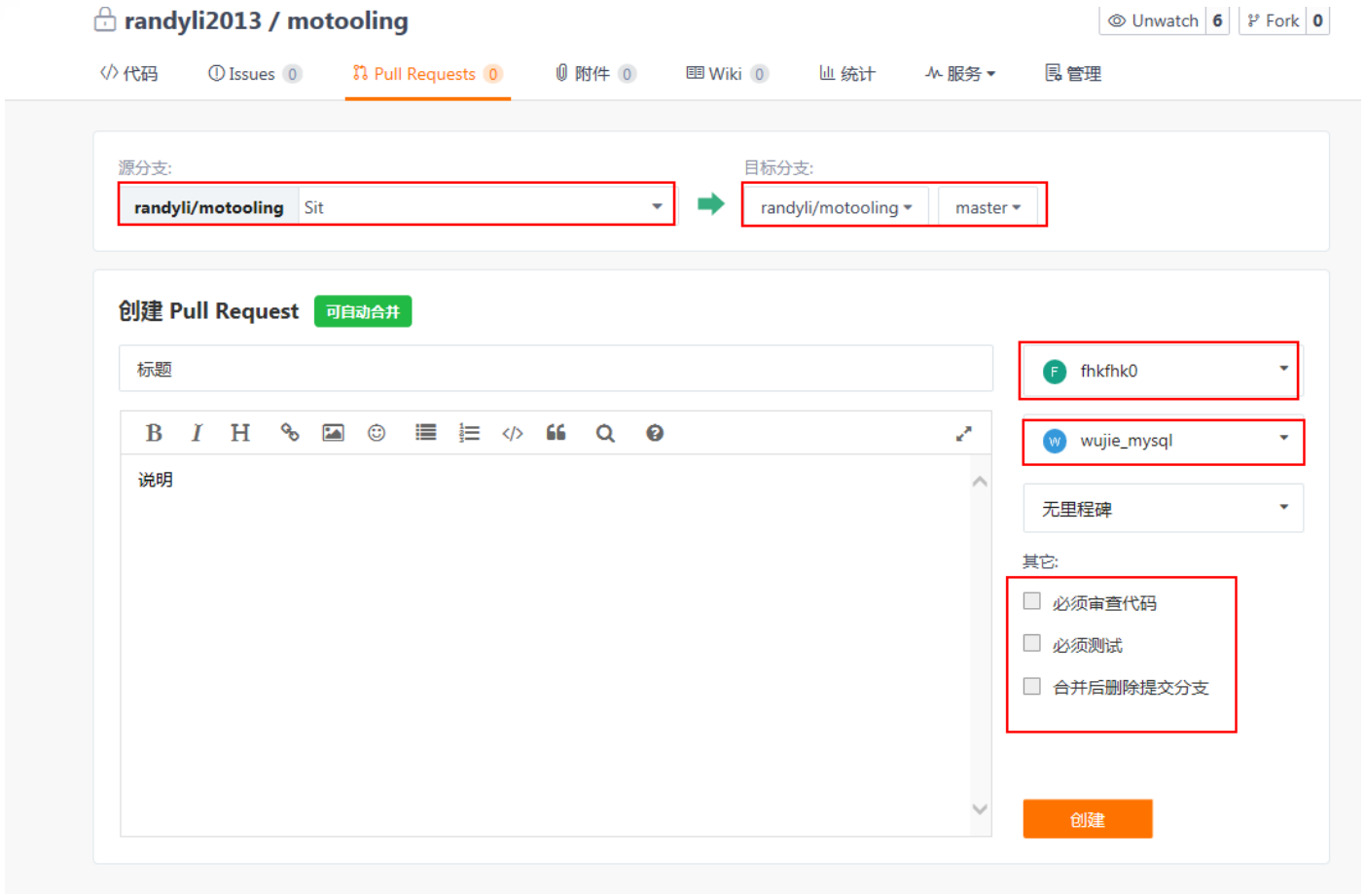
MoWork

**7.开发者和正式仓库做同步**



```
git pull upstream master
```

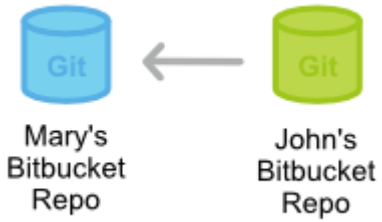
**五、 Pull Requests--是提供的**让开发者更方便地进行协作的功能，提供了友好的**Web界面可以在提**议的修改合并到**正式**项目之前对修改进行讨论。



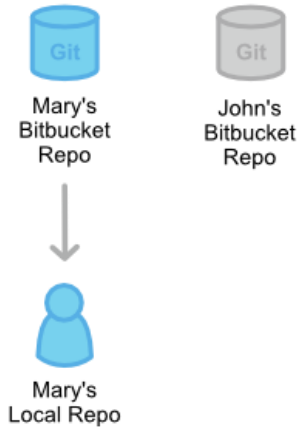

**码云上的Pull Requests**

# 示例

**1.小红fork正式项目**




**2.小红克隆她的Bitbucket仓库**




git clone https://user@bitbucket.org/user/repo.git

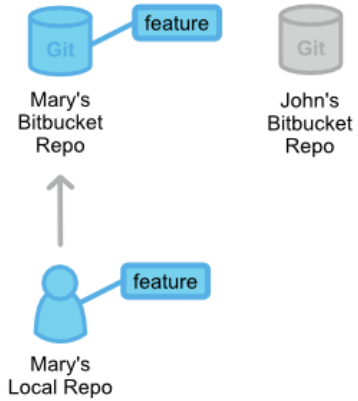
git clone会自动创建origin远程别名，是指向小红fork出来的仓库

MoWork

**3.小红开发新功能**

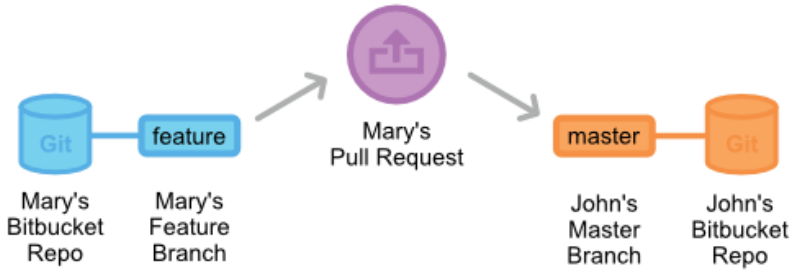

```
git checkout -b some-feature
# 编辑代码
git commit -a -m "Add first draft of some feature"
```

**4.小红push功能到她的Bitbucket仓库中**




```
git push origin some-branch
```

**3.小红发起Pull Requests**



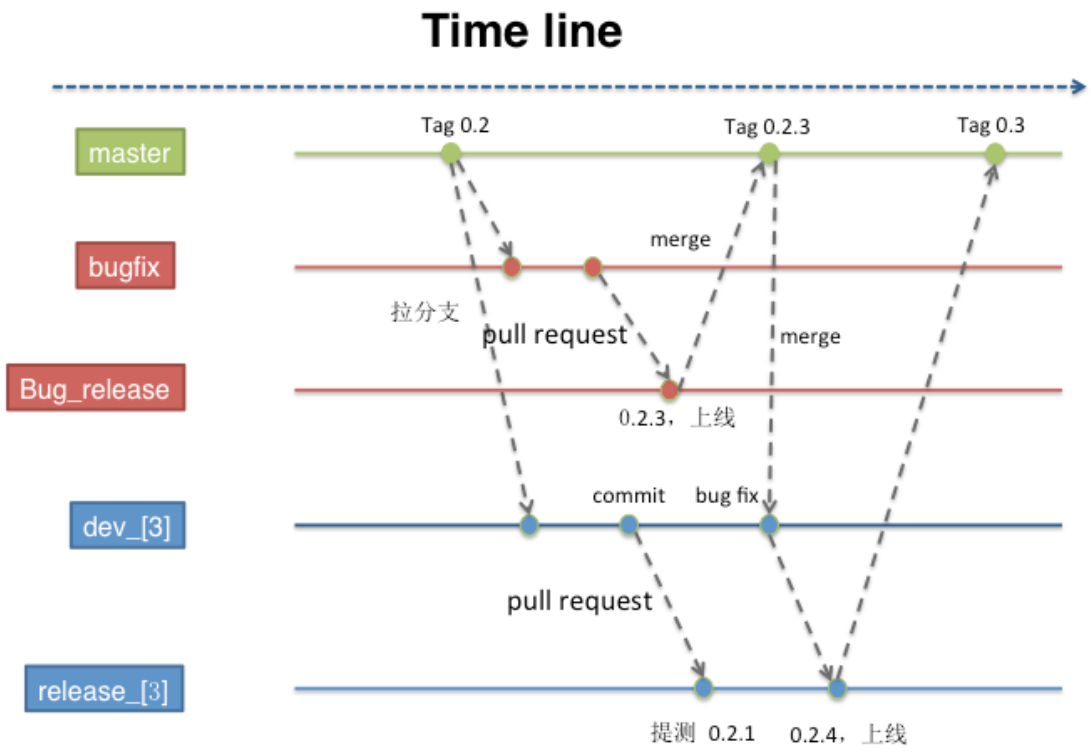

```
git checkout -b some-feature
# 编辑代码
git commit -a -m "Add first draft of some feature"
```

**4.小明review Pull Requests**

**5.小红补加提交**

**6.小明接受Pull Requests， 将其分支merge到master分支上**

# 六、企业日常开发模式




第三次迭代，同时需要修复第二次迭代的线上**bug**