Gitlab 使用教程

# 一、设置 hosts

     Windows用户编辑：c:\windows\system32\drivers\etc\hosts，

     Linux/Mac用户编辑: sudo nano /etc/hosts

     在里面添加：

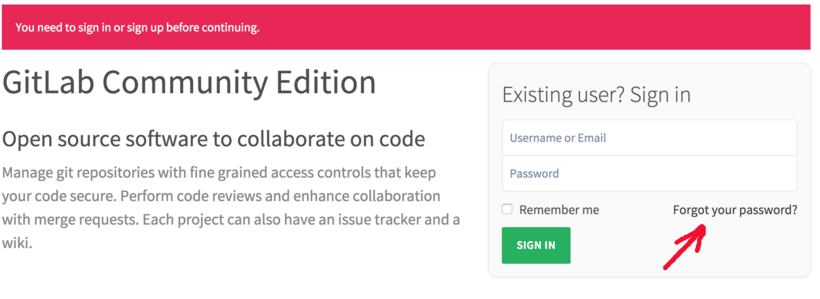
     172.16.5.121 gitlab.vpclub

# 二、设置gitlab登录账户

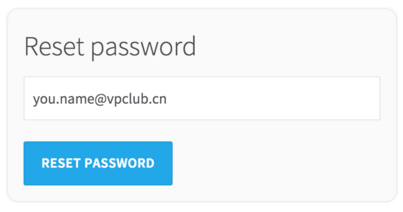
1. 在浏览器输入[http://gitlab.vpclub](http://gitlab.vpclub/):8022



1. 点击 Forgot your password?



1. 输入你的邮箱地址: 如 [your.name@vpclub.cn](mailto:your.name@vpclub.cn)



1. 接下来去邮箱查看你的邮件，你很快会收到来自 Gitlab 的邮件
   * + 1. 点击链接：Change your password
       2. 跳转到gitlab设置好密码后进行下一步
2. 现在你可以登陆 [http://gitlab.vpclub](http://gitlab.vpclub/):8022

# 三、设置密钥与git config

1. 回到操作系统命令提示符下生成ssh密钥及设置git用户信息

Windows:

**$** ssh-keygen -t rsa -b 4096 -C "[your.name@vpclub.cn](mailto:your.name@vpclub.cn)"

**$** git config --global user.name "Your Name"

**$** git config --global user.email "[your.name@vpclub.cn](mailto:your.name@vpclub.cn)"

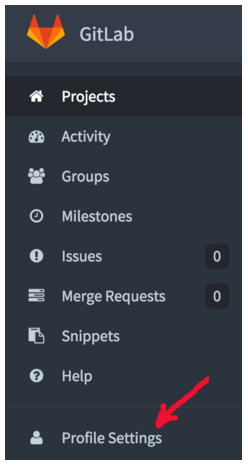
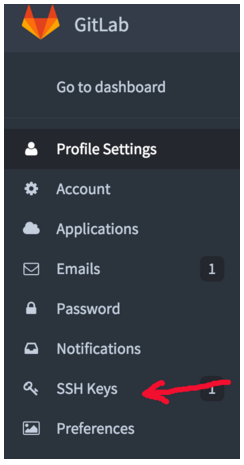
Other:

**$** ssh-keygen -t rss -b 4096 -C "[your.name@vpclub.cn](mailto:your.name@vpclub.cn)"

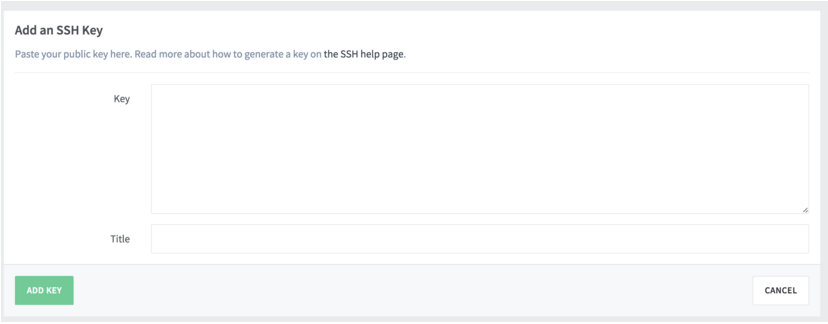
**$** git config --global user.name "Your Name"

**$** git config --global user.email "[your.name@vpclub.cn](mailto:your.name@vpclub.cn)"

1. 登录gitlab后，先点击 Profile Settings >

 ->  -> 

1. 复制刚才生成好的ssh公共密钥 id\_rsa.pub的文本内容并粘贴到下面文本框内：
   1. Windows：c:\Users\yourname\.ssh\id\_rsa.pub;
   2. Linux/Mac: ~/.ssh/id\_rsa.pub )



1. 到现在为止，你已经完成了gitlab的设置，接下来你可以克隆项目到本地进行开发了。

关于git的使用方法请参考：[https://git-scm.com/book/zh/v2](https://git-scm.com/book/zh/v2" \t "_blank)

# 四、常用的git代码分支管理策略

下面是实际开发过程中最常用的git代码分支管理策略。

1. 从development分支创建自己的feature分支，这里取名为 order-detail

git checkout -b order-detial development

1. 用下面命令查看创建的分支

git branch -a

1. 接下来就可以开发了，开发完成后用 git status 查看更改过的文件

git status .

1. 找到代码审查人审查你的代码是否合格

首先运行测试用例给代码审查人确认；

解说逐个文件的更改给审查人审查。

1. 审查没有问题后提交代码到 feature 分支

git add .

git status

git commit

(press i to input below text)

[Description]: Your comments.

[Reviewer]: The name of reviewer

(press esc and :wq to commit )

1. 把提交到代码推到远程服务器

git push origin order-detial

1. 最后到 [http://gitlab.vpclub](http://gitlab.vpclub/) 确认代码是否提交到远程分支

# 五、Git分支管理策略

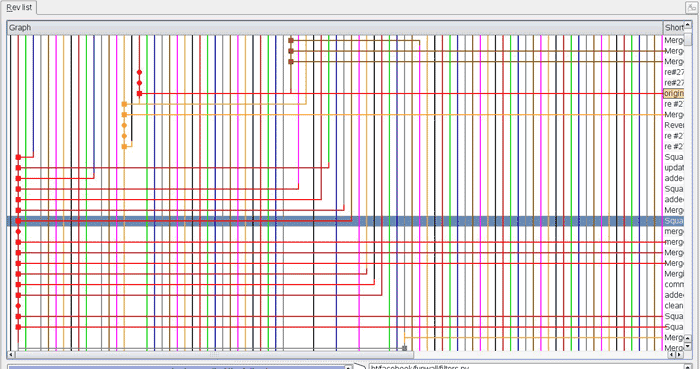
如果你严肃对待编程，就必定会使用"[版本管理系统](http://www.ruanyifeng.com/blog/2008/12/a_visual_guide_to_version_control.html)"（Version Control System）。

眼下最流行的"版本管理系统"，非[Git](http://git-scm.com/" \t "_blank)莫属。



相比同类软件，Git有很多优点。其中很显著的一点，就是版本的分支（branch）和合并（merge）十分方便。有些传统的版本管理软件，分支操作实际上会生成一份现有代码的物理拷贝，而Git只生成一个指向当前版本（又称"快照"）的指针，因此非常快捷易用。

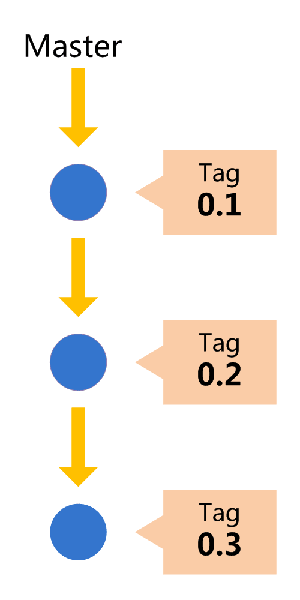
但是，太方便了也会产生副作用。如果你不加注意，很可能会留下一个枝节蔓生、四处开放的版本库，到处都是分支，完全看不出主干发展的脉络。



[Vincent Driessen](http://nvie.com/)提出了一个分支管理的[策略](http://nvie.com/posts/a-successful-git-branching-model/" \t "_blank)，我觉得非常值得借鉴。它可以使得版本库的演进保持简洁，主干清晰，各个分支各司其职、井井有条。理论上，这些策略对所有的版本管理系统都适用，Git只是用来举例而已。如果你不熟悉Git，跳过举例部分就可以了。

**一、主分支Master**

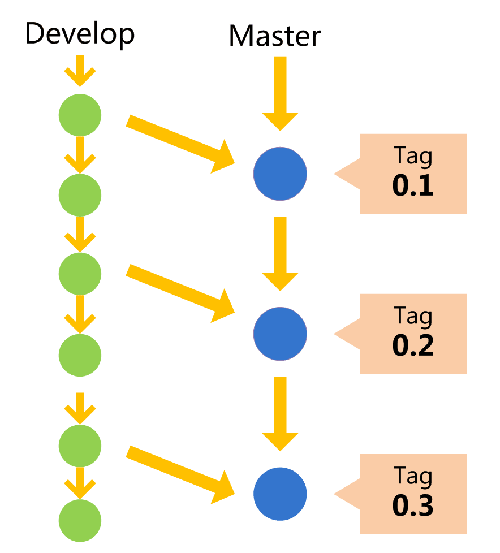
首先，代码库应该有一个、且仅有一个主分支。所有提供给用户使用的正式版本，都在这个主分支上发布。



Git主分支的名字，默认叫做Master。它是自动建立的，版本库初始化以后，默认就是在主分支在进行开发。

**二、开发分支Development**

主分支只用来分布重大版本，日常开发应该在另一条分支上完成。我们把开发用的分支，叫做Development。



这个分支可以用来生成代码的最新隔夜版本（nightly）。如果想正式对外发布，就在Master分支上，对Development分支进行"合并"（merge）。

Git创建Development分支的命令：

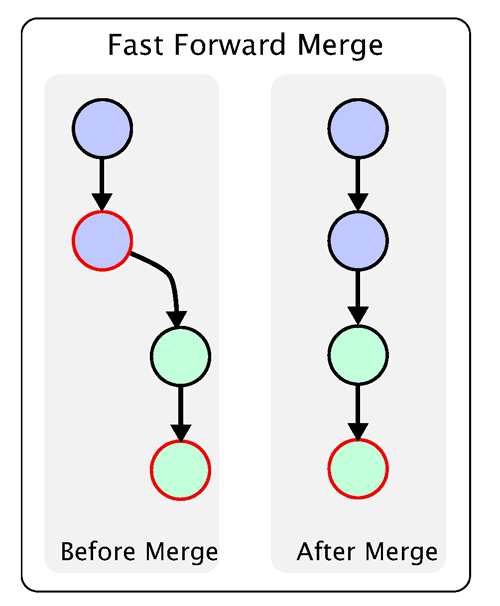
　　git checkout -b development master

将Development分支发布到Master分支的命令：

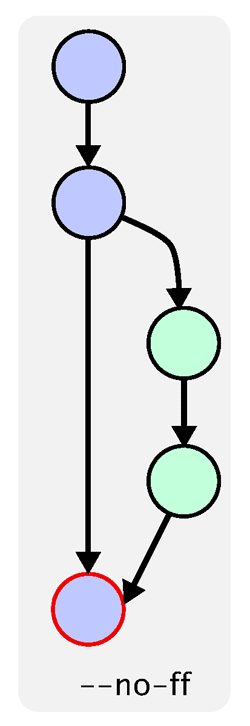
　　# 切换到Master分支  
　　git checkout master

　　# 对Development分支进行合并  
　　git merge --no-ff development

这里稍微解释一下，上一条命令的--no-ff参数是什么意思。默认情况下，Git执行"快进式合并"（fast-farward merge），会直接将Master分支指向Develop分支。



使用--no-ff参数后，会执行正常合并，在Master分支上生成一个新节点。为了保证版本演进的清晰，我们希望采用这种做法。关于合并的更多解释，请参考Benjamin Sandofsky的[《Understanding the Git Workflow》](http://sandofsky.com/blog/git-workflow.html" \t "_blank)。



**三、临时性分支**

前面讲到版本库的两条主要分支：Master和Development。前者用于正式发布，后者用于日常开发。其实，常设分支只需要这两条就够了，不需要其他了。

但是，除了常设分支以外，还有一些临时性分支，用于应对一些特定目的的版本开发。临时性分支主要有三种：

　　\* 功能（feature）分支

　　\* 预发布（release）分支

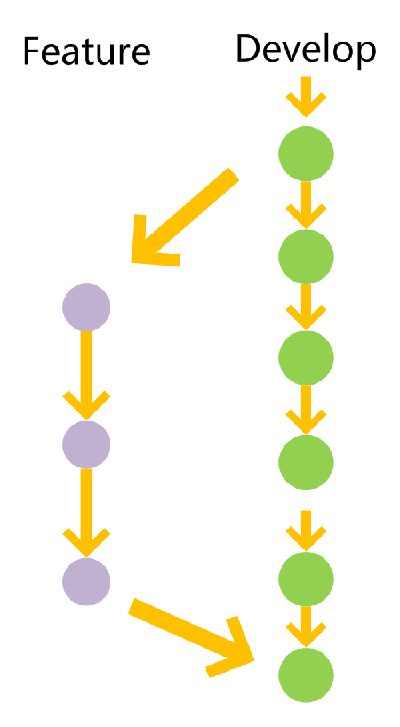
　　\* 修补bug（fixbug）分支

这三种分支都属于临时性需要，使用完以后，应该删除，使得代码库的常设分支始终只有Master和Development。

**四、 功能分支**

接下来，一个个来看这三种"临时性分支"。

第一种是功能分支，它是为了开发某种特定功能，从Development分支上面分出来的。开发完成后，要再并入Development。



功能分支的名字，可以采用feature-\*的形式命名。

创建一个功能分支：

　　git checkout -b feature-x development

开发完成后，将功能分支合并到development分支：

　　git checkout development

　　git merge --no-ff feature-x

删除feature分支：

　　git branch -d feature-x

**五、预发布分支**

第二种是预发布分支，它是指发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。

预发布分支是从Development分支上面分出来的，预发布结束以后，必须合并进Development和Master分支。它的命名，可以采用release-\*的形式。

创建一个预发布分支：

　　git checkout -b release-1.2 development

确认没有问题后，合并到master分支：

　　git checkout master

　　git merge --no-ff release-1.2

　　# 对合并生成的新节点，做一个标签  
　　git tag -a 1.2

再合并到development分支：

　　git checkout development

　　git merge --no-ff release-1.2

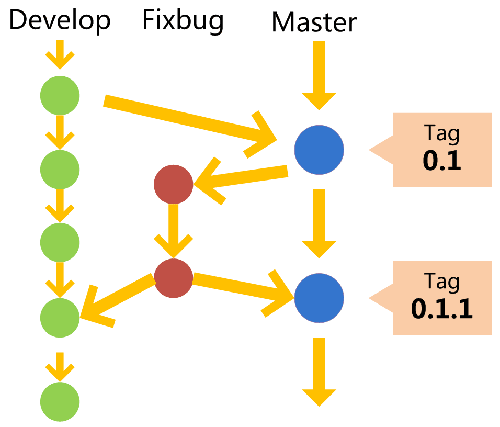
最后，删除预发布分支：

　　git branch -d release-1.2

**六、修补bug分支**

最后一种是修补bug分支。软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。

修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和Development分支。它的命名，可以采用fixbug-\*的形式。



创建一个修补bug分支：

　　git checkout -b fixbug-0.1 master

修补结束后，合并到master分支：

　　git checkout master

　　git merge --no-ff fixbug-0.1

　　git tag -a 0.1.1

再合并到development分支：

　　git checkout development

　　git merge --no-ff fixbug-0.1

最后，删除"修补bug分支"：

　　git branch -d fixbug-0.1