Git 工作流

> Author：Marlous（整理、编写）
>
> E-mail：goonecat@foxmail.com

> Date：2018/10/23

一 两种开发合作模式

参考：参考问题回答

1. 属于主仓库团队项目贡献者一员，大家使用同一个仓库进行合作开发。分支开发功能，开发完毕，建立 pull request，进行 code review，最终合并到主仓库 dev 分支。
2. 不是主仓库团队项目贡献者一员，则必须大家 fork 仓库。开发完毕后，从自己的远端仓库建立 pull request 到主仓库的 dev 分支，由管理代码的人进行合并。

二 概述 Github flow

1. 创建一个新分支。
   同一时刻多个想法，创建分支来进行管理。
2. 添加新版本。
   不断做新功能出来。
3. 开启一个 pull request。
   发起对做的各个版本的讨论，代码审核。
4. 合并分支，然后部署。
   大家审核了 pull request 并通过测试，后合并到主分支上。

三 详细 Git flow

1. 项目管理者创建 master 分支，在 master 分支上创建 dev 分支。
2. 任务分配给不同的人员（不同特性）：

- 不同人员克隆主仓库到本地，在克隆的仓库的 dev 分支上创建自己的特性分支（或 fork 此项目后，克隆到本地，在其 dev 分支上创建自己的特性分支）。
- 然后可以提交自己的分支到远端主仓库备份，让其他成员知道进度，远程跟踪分支 `git push -u origin feature`（或提交自己的分支到自己 fork 的远端主仓库）。

3. 进行自己特性分支上的开发。然后 commit。
4. 更新、合并、推送：

- 从远端主仓库获取，更新本地的 dev 分支，注意区别，`git pull` = `git fetch + git merge(rebase)`；
- 可能要处理冲突；
- 处理完冲突 commit，将更新后的本地 dev 分支 merge 到自己的任务分支，得到要推送的分支。也可以用 rebase 来将更新的本地 dev 分支与自己的特性分支合并，得到要推送的分支（rebase 和 merge 效果一样，

但让之前的分支提交记录消失，让分支线看起来更清晰）。
- 需 code review：发起 pull request，然后进入第 5 步；或等管理者对代码直接 review，然后推送到远端与目标分支合并，结束。
  不需要 code review：（如个人项目）直接将特性分支与远端目标分支合并，结束。

5. 团队需要 pull request，管理者收到 pull request：

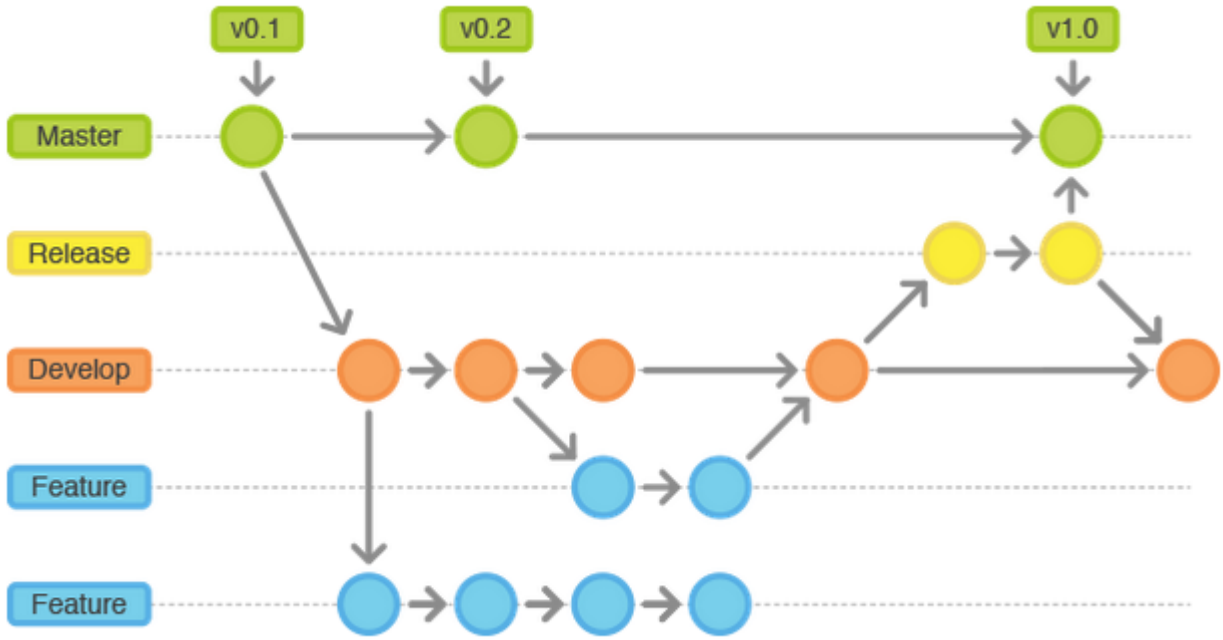
- 不同仓库合作开发：获取 patch 文件，或从远端分支关联获取；同仓库则省略此步。
- 在 dev 分支（将要被合并的分支）上创建并切换到临时分支。
- 将请求合并分支与临时分支合并，进行测试。
- 测试没问题后删除临时分支，接受 pull request。
- 有问题则在 pull request 中讨论和解决问题，要求请求者解决，或管理者在临时分支上测试并解决后将此临时分支与被请求合并的分支合并，删除临时分支。

6. 注：在开始下个任务前或任务中需注意更新自己本地的 dev 分支。

四 各分支使用

> GitFlow：git 的最佳实践。
>
> 项目地址：https://github.com/nvie/gitflow
>
> 文章：https://nvie.com/posts/a-successful-git-branching-model/

git 的最佳实践简化图：



1. master 分支作为中央仓库主分支，并且功能分支不是从 master 分支上拉出新分支，而是使用 dev 分支作为父分支。
2. 一旦 develop 分支上有了做一次发布（或者说快到了既定的发布日）的足够功能，就从 develop 分支上 fork 一个发布分支，即 release。新建的分支用于开始发布循环，所以从这个时间点开始之后新的功能不能再加到这个分支上,这个分支只应该做 Bug 修复、文档生成和其它面向发布任务。一旦对外发布的工作都完成了，发布分支合并到 master 分支并分配一个版本号打好 Tag。另外，这些从新建发布分支以来的做的修改要合并回 develop 分支。
3. 预发布分支命名: release-

4. 维护分支或说是热修复（hotfix）分支用于生成快速给产品发布版本（production releases）打补丁，这是唯一可以直接从 master 分支 fork 出来的分支。修复完成，修改应该马上合并回 master 分支和 dev 分支（当前的发布分支），master 分支应该用新的版本号打好 Tag。

五 实践（从零到一）

主开发者创建项目

- 注册 GitHub 账号，安装 Git。
- 本地设置，Git 中设置可以登陆到 GitHub：

```
# 设置用户名、邮箱
$ git config --global user.name "you account name"
$ git config --global user.email youemail@example.com

# 生成公钥
ssh-keygen -t rsa -C "youemail@example.com"
ssh git@github.com  //测试是否成功。
```

- 将公钥添加至自己的 GitHub 中。
- 在远端设置好项目仓库。
- 本地初始化，关联、获取合并、推送。

```
git init

git remote add origin git@server-name:path/repo-name.git

git pull origin master

git push origin master
```

- 切换并创建分支：`git checkout -b [name]`，创建某分支，然后 push。

参与开发者

- fork 或 clone 项目到本地：`git clone git://github.com/User/project.git`（项目链接）
- 在 dev 分支上，切换创建自己的特性分支：`git checkout -b myfeature`
- 备份自己的特性分支：
  可以提交自己的分支到远端主仓库备份，让其他成员知道进度，远程跟踪分支 `git push -u origin feature`。
  或提交自己的分支到自己 fork 的远端主仓库 `git push -u origin feature`。
  （如果当前分支只有一个追踪分支，那么主机名都可以省略。git push，默认只推送当前分支。 -u 为指定 origin 为默认主机，之后可直接使用 git push。）
- 在特性分支上做开发工作。

```
# 开发完后提交
git add -A
git commit -m "fixed"  # 可以添加一些注释
```

- 更新本地 dev 分支；解决冲突；合并到自己的分支。

```
# 更新本地 dev 分支（大家往上合并功能的分支）
git checkout dev
git fetch origin dev
解决冲突。
git merge origin/dev

# 更新后的 dev 合并到自己的特性分支
git checkout feature   # 切换到自己的特性分支上。
git merge dev   # 合并 dev 到自己的特性分支。 merge 由三种参数方式；或使用 git rebase dev。
```

- 发 pull request。或直接推送合并：`git push origin dev`

主开发者测试、合并到主仓库某（dev）分支

- 先创建测试用的临时分支：`git checkout -b PRx` （x 为 pull request 号）
- 两种方法获取来测试、合并。参考文章：

1. 下载 PR 生成的 patch 文件，合并进临时分支。

```
# 获取 pull request 的 patch，-L 参数表示如果有 302 跳转，curl 会自动跟进；pull/8 表示该仓库收到的第 8 个 PR。
curl -L http://github.com/User/project/pull/8.patch | git am
或直接下载某一个 commit 即可：
curl https://github.com/User/project/commit/bd770141029f49bcfa2e0d6e6e6282b531e69179.patch | git am

# 将请求合并的分支与自己的临时分支合并
git checkout PRx
git merge 请求的分支名
```

2. 为 PR 创建一个远程分支，追踪提交者的仓库。合并进临时分支。

```
git remote add 远程仓库名 git@server-name:path/repo-name.git  # 为区分远程仓库名不用 origin，取个其它名字。

git fetch 远程仓库名 请求的分支名

# 将请求合并的分支与自己的临时分支合并
git checkout PRx
git merge 请求的分支名
```

- 测试成功，接受合并到远端主仓库。
  或修复后将此临时分支（PRx）推送到主仓库：`git push origin master`。

准备下轮特性开发

- 让本地主分支代码最新：

```
git fetch origin master  #下载一下远程代码
gti checkout master
git merge orgin/master
```

六 总结

拉代码 ->（创建特性分支/临时分支） -> 开发和更新代码 ->（合并进特性分支/临时分支） -> 推送代码

- 克隆仓库代码到本地，在此基础上创建特性分支（可设置追踪分支）。
- 完成特性的开发做成版本。
- 更新本地 dev 分支（远程 dev 合并到本地 dev）并解决冲突；然后将其合并到自己的特性分支上（或 rebase 更新后的本地 dev）。
  注：可以直接从远程 dev 分支获取更新合并到本地特性分支，不需要本地 dev 分支中转。
- 发 pull request，或直接推送到远端主仓库。
- 接受 pull request 前（没问题或请求者修完，直接接受 pull request），先创建临时分支，将获取的补丁代码合并进临时分支。
- 测试完后，将此临时分支推送到远端主仓库，删除临时分支。