

基于git的分支开发流程简介

- [参考文档1](#)
- [参考文档2](#)
- [参考文档3](#)
- [参考文档4](#)
- [参考文档5](#)
- [参考文档6](#)
- [参考文档7](#)

简介

Git Flow 是 Vincent Driessen 提出了一个分支管理的策略，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。

Git Flow 常用分支

- Master 分支 也就是我们经常使用的Production分支，这个分支最近发布到生产环境的代码，最近发布的Release，这个分支只能从其他分支合并，不能在这个分支直接修改
- Develop 分支 这个分支是我们的主开发分支，包含所有要发布到下一个Release的代码，这个分支主要合并自其他分支，比如Feature分支
- Feature 分支 这个分支主要是用来开发一个新的功能，一旦开发完成，我们合并回Develop分支进入下一个Release
- Release分支 当你需要一个发布一个新Release的时候，我们基于Develop分支创建一个Release分支，完成Release后，我们合并到Master和Develop分支
- Hotfix分支 当我们在Production发现新的Bug时候，我们需要创建一个Hotfix, 完成Hotfix后，我们合并回Master和Develop分支，所以Hotfix的改动会进入下一个Release

分支详解

Master 分支

- Master分支不允许直接提交代码，只能从其他分支合并，并且每次合并都要打Tag

Develop 分支

- 最新的开发分支，包含最新完成的功能代码
- 准备Release发布的时候，所有新的功能不允许合并到这个分支，直到发布版本完成

Feature 分支

- 分支命名 feature/* 如：feature/watch-match, feature/bet-group
- Feature基于Develop分支创建，完成后必须合并回Develop分支
- 合并完分支后一般会删点这个Feature分支，也可以保留

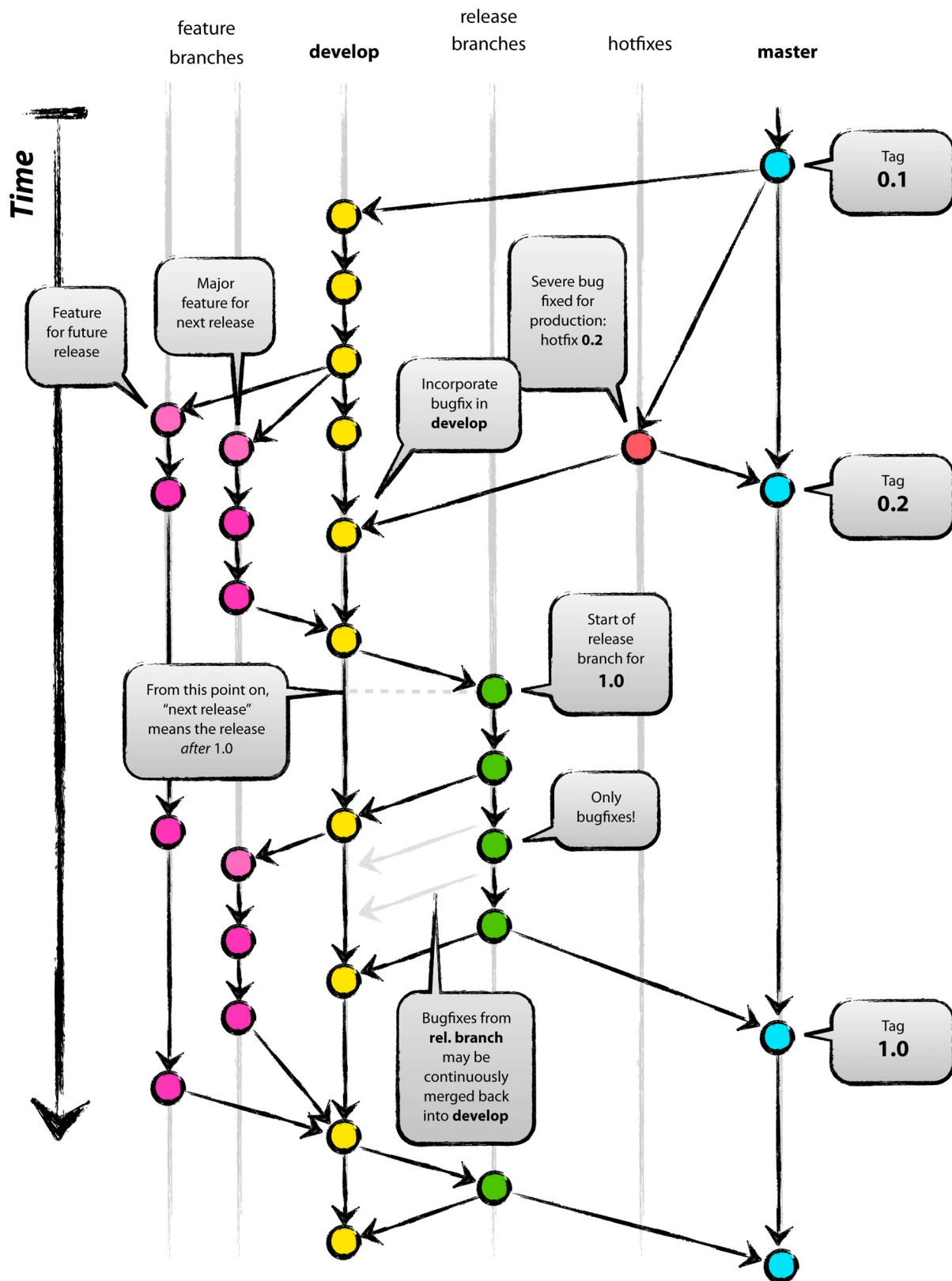
Release 分支

- Release分支基于Develop分支创建，打完Release分支之后，可以在这个Release分支上测试，修改Bug等
- 同时，其它开发人员可以断续开发新的Feature
- 一旦打了Release分支之后，不要从Develop分支上合并新的改动到Release分支

Hotfix 维护分支

- hotfix分支基于Master分支创建，开发完后需要合并回Master和Develop分支
- 同时在Master上打一个tag升级一个小版本号

Git Flow的流程图



注意:

- 本地和远程的同名分支 (develop, origin/develop)，尽量使用 `git rebase` 来代替 `git merge`，如pull时使用 `git pull --rebase origin develop`
- 合并分支时尽量不要使用快速合并，由于默认就是快速合并，所以合并时使用 `--no-ff` 参数。 `git merge --no-ff xxxx`

Git Flow 开发流程代码示例

- 创建 develop 分支

```
# 创建develop分支
git branch develop

# 推送develop分支到远端服务器
git push -u origin develop
```

注意：上面的推送操作一般只由开发组组长一人操作，其他开发者执行如下命令

```
# 如果代码已clone完成可以不执行此步操作
git clone xxxx

# 跟踪develop分支
git checkout -b develop origin/develop
```

- 开始新Feature开发

```
# 基于develop分支创建feature分支
git checkout -b feature/some-feature develop

# 可以选择性推送feature分支到远端，
git push -u origin feature/some-feature

# 开发新功能，修改文件
# 查看文件状态
git status

# 添加代码文件到git版本库
git add some-file
git commit
```

注意：推送feature分支到远端，有如下好处

- 这样可以防止本地代码因其他原因导致丢失
- 当有人和你一起开发一个功能时，可以推送到远端，让别人也可以拉取这个功能分支，和你一起开发功能

- 完成Feature

```
# 拉取远端develop分支合并到当前feature分支
# 如果有冲突，需要解决冲突后提交

# 可以使用如下方法
# git checkout develop
# git pull
# git checkout feature/some-feature
# git rebase develop
# 如果出现冲突，解决冲突后只需要git add
# 不需要commit，然后执行以下命令：
# git rebase --continue

# 也可以使用如下方法合并
# git pull origin develop
git pull --rebase origin develop

# 合并feature分支到develop并推送到远端
git checkout develop
git merge --no-ff feature/some-feature
git push origin develop

# 删除feature分支
git branch -d feature/some-feature

# 选择性的删除远端feature分支
git push origin --delete feature/some-feature
```

- 开始Release

```
# 基于develop分支创建release分支
git pull --rebase origin develop
git checkout -b release/v0.1.0 develop

# 选择性的推送到远端
git push -u origin release/v0.1.0

# 在release/v0.1.0分支上修改bug提交并测试
```

注意：发布前需要确认本次要发布的功能分支都已合并并推送到了远端仓库

- 完成Release

```
# 合并到master分支
git checkout master
git merge --no-ff release/v0.1.0
git push

# 合并到develop分支
git checkout develop
git merge --no-ff release/v0.1.0
git push

# 删除release分支
git branch -d release/v0.1.0

# 删除远程分支
git push origin --delete release/v0.1.0

# 在master分支打tag并推送
git tag -a v0.1.0 master
git push --tags
```

- 开始Hotfix

```
# 拉取最新的master
git checkout master
git pull --rebase origin master

# 从master分支创建hotfix分支并检出
git checkout -b hotfix/v0.1.1 master

# 修改代码，修复bug
```

- 完成Hotfix

```
# 合并到master分支
git checkout master
git merge --no-ff hotfix/v0.1.1
git push

# 合并到develop分支
git checkout develop
git merge --no-ff hotfix/v0.1.1
git push

# 删除分支
git branch -d hotfix/v0.1.1

# 在master分支打tag并推送
git tag -a v0.1.1 master
git push --tags
```

使用 Git flow 工具

- [参考文档1](#)
- [参考文档2](#)
- [参考文档3](#)

安装

- OS X

```
brew install git-flow
```

- Linux

```
yum install gitflow
#apt-get install git-flow
```

- Windows

```
# 安装git之后自带git-flow
```

使用实例

- 初始化

```
# 初始化设置一些基本参数
git flow init

# 推送 develop 分支到远程仓库
# 只需要开发组组长一个人操作即可
git push origin develop
```

- 开始新Feature开发

```
# 开发feature分支
# 开始之前可以先更新一下远端develop分支的最新代码
git checkout develop
git pull --rebase origin develop
git flow feature start some-feature

# 选择性的推送到远端
git flow feature publish some-feature
```

其他可能会用到的命令

```
# 获取一个远端的feature分支
git flow feature pull some-other-feature

# 跟踪在origin上的一个feature分支
git flow feature track some-other-feature
```

- 完成Feature开发

```
# 修改编写代码完成之后完成
git flow feature finish some-feature

# 拉取远端的develop分支合并到本地分支
git pull --rebase origin develop

# 如果出现冲突，解决冲突后只需要git add
# 不需要commit，执行以下命令：
# git rebase --continue

# 推送develop分支到远端
git push origin develop
```

- 开始Release

当你认为现在在 develop 分支的代码已经是一个成熟的 release 版本时，这意味着：

- 它包括所有新的功能和必要的修复；

- 它已经被彻底的测试过了。如果上述两点都满足，那就是时候开始生成一个新的 release 了

```
# 拉取最新的develop分支
git checkout develop
git pull --rebase origin develop

# 冲突解决办法同上

git flow release start v0.1.0

# 选择性的推送release 分支
git flow release publish v0.1.0
```

可能会用到的其他命令

```
# 签出 release 版本的远程变更
git flow release track v0.2.0
```

- 完成Release

这个命令会完成如下一系列的操作：

- 首先，git-flow 会拉取远程仓库（把远程develop分支合并到当前分支），以确保目前是最新的版本。
- 然后，release 的内容会被合并到 master 和 develop 两个分支中去，这样不仅产品代码为最新的版本，而且新的功能分支也将基于最新代码。
- 为便于识别和做历史参考，release 提交会被标记上这个 release 的名字
- 清理操作，版本分支会被删除，并且回到 develop

```
# 完成
git flow release finish v0.1.0

# 推送develop分支
git pull --rebase origin develop
git push origin develop

# 推送master分支
git pull --rebase origin master
git push --tags origin master
```

- 开始Hotfix

```
# 拉取最新的master, develop分支
git checkout master
git pull --rebase origin master
git pull -t
#git checkout develop
#git pull --rebase origin develop
git flow hotfix start v0.1.1

# 选择性的推送
git flow hotfix publish v0.1.1
```

可能会用到的其他命令

```
# 跟踪远端的分支
git flow hotfix track v0.1.2
```

- 完成Hotfix

```
# 完成
git flow hotfix finish v0.1.1

# 推送master分支
git pull --rebase origin master
```

```
git push --tags origin master
```

```
# 推送develop分支
git pull --rebase origin develop
git push origin develop
```

整合gitlab/gogs使用

虽然设置master和develop为保护分支，不允许直接推送代码，但是也可以设置为允许开发组组长推送代码。这样在合并到 master和develop 分支的时候，就可以不用发起pull request 在web页面合并代码，而直接让开组组长合并到 master和develop 分支，然后推送 develop和master 分支到远端仓库。

- 在gogs/gitlab上创建好相应项目的仓库并添加相应的人员，并设置好相应的权限，开发组组长创建develop分支，推送到远程仓库。并设置master和develop为保护分支，不允许直接推送代码，只允许创建合并请求合并代码到这两个分支。

```
git clone ssh://git@dev.famulei.com:10022/will/git-flow.git
cd git-flow
git branch develop
git push origin develop
```

使用git flow操作方式

```
git clone ssh://git@dev.famulei.com:10022/will/git-flow.git
cd git-flow
git flow init
git push origin develop
```

- 开发人员基于develop分支创建自己的功能分支，并把功能分支推送到远程仓库

```
git clone ssh://git@dev.famulei.com:10022/will/git-flow.git
cd git-flow
# 如果已经clone过远程仓库，也可直接使用 git pull更新即可

# 创建功能分支，开发功能
git checkout -b develop origin/develop
git checkout -b feature/add-match develop

# 如果是多人配合开发这个功能的话，推送到远程仓库
git push -u origin feature/add-match

# 修改代码，添加好功能。可以不定期的推送功能分支到远程，防止代码因其他原因丢失
# 自己测试基本没有bug问题之后，推送更新到远程仓库，为下一步合并做准备

# 推送之前先把远程develop合并到当前开发的功能分支
# 防止推送功能分支到远端合并到develop分支时出现冲突，无法正常合并
# 如下的操作在开发功能的过程中也可以经常的执行，
# 避免最后推送合并时有太多的冲突要解决
git pull --rebase origin develop

# 如果上条命令出现冲突，解决冲突后不需要 commit，只需要 add 即可
git add .
git rebase --continue

# 推送到远程仓库
git push -u origin feature/add-match
```

使用git flow操作方式

```
git clone ssh://git@dev.famulei.com:10022/will/git-flow.git
cd git-flow
git flow init
git flow feature start add-match
# 修改代码，添加功能

# 合并远程仓库develop分支
git pull --rebase origin develop

# 推送
```

```
git flow feature publish add-match
```

- 发起pull request合并请求，请求把自己的功能分支合并进develop分支，开发组查看代码， 审查之后，同意合并请求，并删除该功能分支，代码合并进develop。之后进行功能测试。

```
# 合并进develop分支后可以删除本地的功能分支
git branch -d feature/add-match
```

使用git flow操作方式

```
git flow feature finish add-match
```

发起合并请求，写清楚描述信息

对比文件变化

对比两个分支间的文件变化并发起一个合并请求。

基准分支: develop ▼

...

对比分支: feature/1 ▼

添加功能1

内容编辑

效果预览

添加功能1

- 添加比赛

- 查看比赛

- 结束比赛

拖曳文件到此处或单击上传

创建合并请求

标签 ⚙

未选择标签

里程碑 ⚙

未选择里程碑

指派成员 ⚙

未指派成员

组长和开发人员可以在如下页面查看合并信息，文件修改信息，代码提交的信息。 确认没有问题后，同意合并请求，并删除功能分支，也可以根据情况保留。

Page 8/11

Monday, May 22, 2017, 2:02 PM

#1 添加功能1


编辑

🔔 开启中 will 请求将 1 次代码提交从 will/feature/1 合并至 will/develop

💬 对话内容 1

🔗 代码提交 1


📄 文件变动 1



will 评论于 1 分钟之前

添加功能1


- 添加比赛
- 查看比赛
- 结束比赛



will 评论于 9 秒之前

这个代码写的还不错，可以合并

所有者 ✎ ×



✓ 该合并请求可以进行自动合并操作。

🔗 合并请求

标签 ⚙️

未选择标签


里程碑 ⚙️

未选择里程碑

指派成员 ⚙️

未指派成员

1 名参与者



- 当下次要上线的功能都已经开发完成，并且都已经合并进develop分支之后，之后开发的新功能不允许再合并到develop分支，直到这次上线完成之后。开发组组长创建release分支，并把release分支推送到远程仓库

```
git checkout develop
git pull --rebase origin develop
git checkout -b release/v0.1.0 develop
git push -u origin release/v0.1.0
```

登录gitlab/gogs将该分支设置为保护分支

使用git flow操作方式

```
git checkout develop
git pull --rebase origin develop
git flow release start v0.1.0
git flow release publish v0.1.0
```

- 全面测试发现bug后直接在release分支修改代码，或者基于release创建新的分支修改代码，修改完成之后，合并进release分支。

```
# 开发人员更新代码，并创建本地分支
git checkout develop
git pull --rebase origin develop
git checkout -b release/v0.1.0 develop
git pull --rebase origin release/v0.1.0

# 基于发布分支创建bug修复分支
git checkout -b fixbug-v0.1.0/bug1 release/v0.1.0

# 修复bug提交测试通过后准备推送到远程仓库
# 推送之前先把release/v0.1.0合并到bug分支
# 防止推送bug修复分支到远端合并到release/v0.1.0分支时出现冲突，无法正常合并
git pull --rebase origin release/v0.1.0

# 如果上条命令出现冲突，解决冲突后不需要 commit，只需要 add 即可
git add .
git rebase --continue

# 推送到远端仓库
git push -u origin fixbug-v0.1.0/bug1

# 发起pull request合并请求合并到release/v0.1.0分支
```

如果出现冲突无法正常合并时，使用如下解决方法

```
# 合并release/v0.1.0分支到自己的bug分支解决冲突
git checkout fixbug-v0.1.0/bug5

# 解决冲突
git pull --rebase origin release/v0.1.0
# 如果上条命令出现冲突，解决冲突后不需要 commit，只需要 add 即可
git add .
git rebase --continue

# 推送到远端仓库
git push -u origin fixbug-v0.1.0/bug5

# 之后合并请求就可以正常合并了
```

- 当确定bug都已经清除，确认可以上线之后，发起pull request合并请求，开发组组长做如下操作，合并进master和develop分支。

```
# 更新 release 分支
git checkout release/v0.1.0
git pull --rebase origin release/v0.1.0

# 拉取 develop 分支更新到当前分支
git pull --rebase origin develop
# 如果有冲突和之前一样解决冲突

# 推送
git push origin release/v0.1.0

# 发起合并请求，把release/v0.1.0分支合并到develop和master分支

# 删除本地的发布分支
git branch -d release/v0.1.0
```

使用git flow操作方式

```
git checkout release/v0.1.0
git pull --rebase origin release/v0.1.0

# 推送
git push origin release/v0.1.0

git flow release finish v0.1.0
```

- 在gogs/gitlab发布新版本，开发人员更新代码

```
# 更新master分支
git checkout master
git pull --rebase origin master
git pull -t

# 更新develop分支
git checkout develop
git pull --rebase origin develop
```

- 修复线上bug，当上线之后，发现线上有小bug或者做简单的代码改动。

如果bug较多，也可以基于hotfix分支现创建其他分支，然后申请合并到hotfix分支，最后一起 合并到 develop 和 master分支

```
# 更新代码
git checkout -b hotfix/v0.1.1 master
# 修改代码 测试没有问题之后

# 合并master代码到当时分支
git pull --rebase origin master
# 如果有冲突和之前一样解决冲突

# 推送
git push origin hotfix/v0.1.1
```

```
# 发起合并请求，合并到 master和develop分支  
  
# 重新发布版本增加小版本号  
  
# 删除本地下的hotfix分支  
git branch -d hotfix/v0.1.1
```

使用git flow操作方式

```
# 更新master分支  
git checkout master  
git pull --rebase origin master  
  
# 更新develop分支  
git checkout develop  
git pull --rebase origin develop  
  
git flow hotfix start v0.1.1  
# 修改代码  
git pull --rebase origin master  
git flow hotfix publish v0.1.1  
git flow hotfix finish v0.1.1  
  
# 发起合并请求，合并到 master和develop分支  
  
# 重新发布版本增加小版本号
```