git流程规范

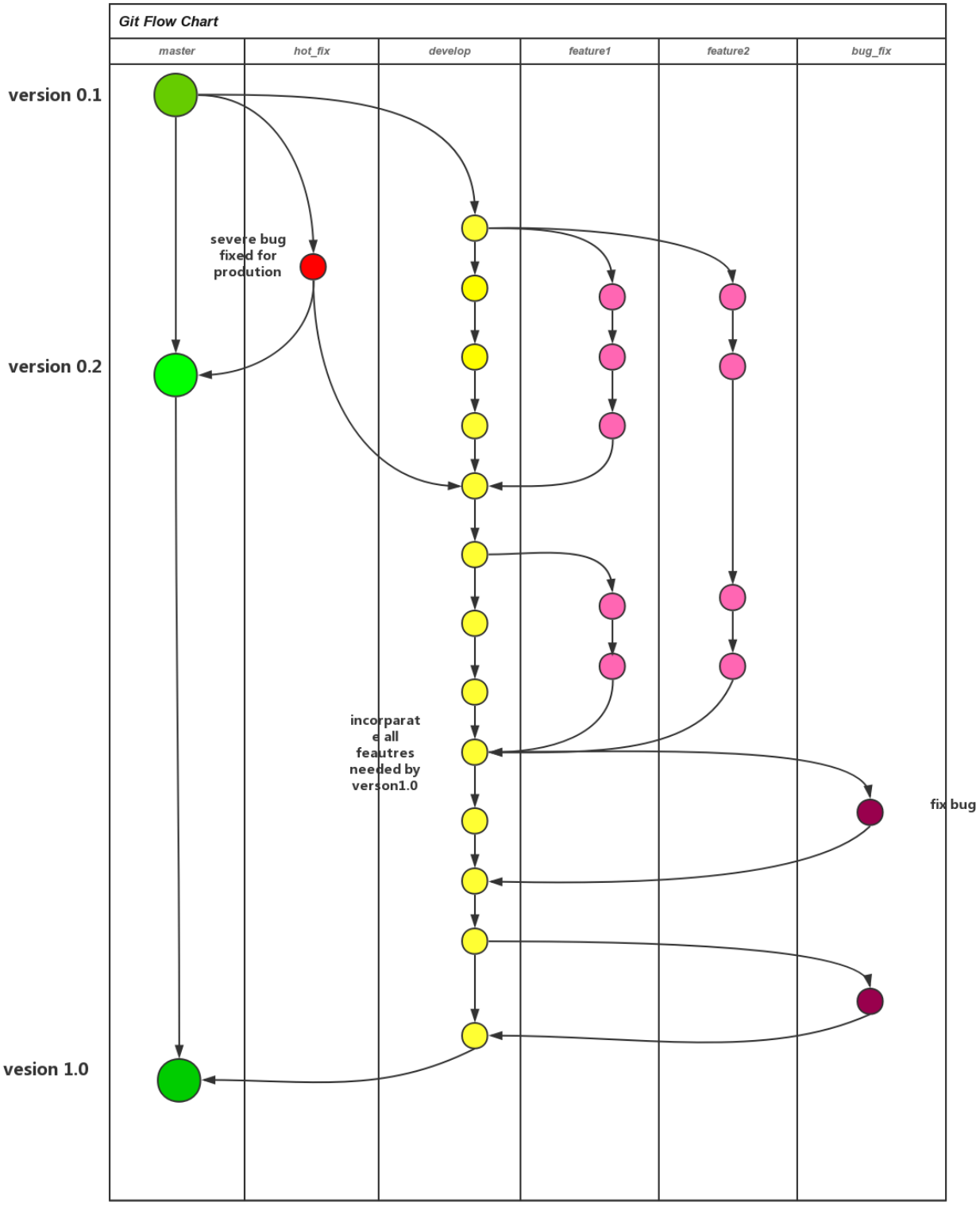
Contents

[TOC]

版本信息

作者	日期	版本	描述
林能刚	2018-05-02	1.0	git流程规范草案
林能刚	2018-06-06	1.1	增加commit message提交规范

1 Git Flow



1.1 master branch

master分支上存放的应该是随时可供在生产环境中部署的代码（Production Ready state）。当开发活动告一段落，产生了一份新的可供部署的代码时，master分支上的代码会被更新。同时，每一次更新，最好添加对应的版本号标签（TAG）

1.2 develop branch

develop分支是保存当前最新开发成果的分支。

当develop分支上的代码已实现了软件需求说明书中所有的功能，通过了所有的测试后，并且代码已经足够稳定时，就可以将所有的开发成果合并回master分支了。对于master分支上的新提交的代码建议都打上一个新的版本号标签（TAG），供后续代码跟踪使用。

因此，每次将develop分支上的代码合并回master分支时，我们都可以认为一个新的可供在生产环境中部署的版本就产生了。通常而言，“仅在发布新的可供部署的代码时才更新master分支上的代码”是推荐所有人都遵守的行为准则。

1.3 feature branch

使用规范： 从develop创建新feature分支，代码必须合并回develop分支。 feature分支的命名可以使用如feature-xxx，通常是在开发一项新的软件功能的时候使用，这个分支上的代码变更最终合并回develop分支。

一般而言，feature分支代码可以保存在开发者自己的代码库中而不强制提交到主代码库里。

1.4 hotfix branch

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develop分支（当前的发布分支），master分支应该用新的版本号打好Tag。

1.5 bugfix branch

修复bug的分支， 每修复一个bug都应切换到新的bugfix branch， 修复成功后再合入develop分支。

Back to TOC

2 git主要操作

我们公司用的git仓库管理系统是GitLab 7.10.0， 使用Git作为代码管理工具，并在此基础上搭建起来的web服务。

2.1 配置个人信息

配置用户名称和电子邮件地址， 这里配上个人在我司gitlab上的账号和邮箱即可， 没有找相关人员申请。

```
$ git config --global user.name alan $ git config --global user.email alan@gmail.com
```

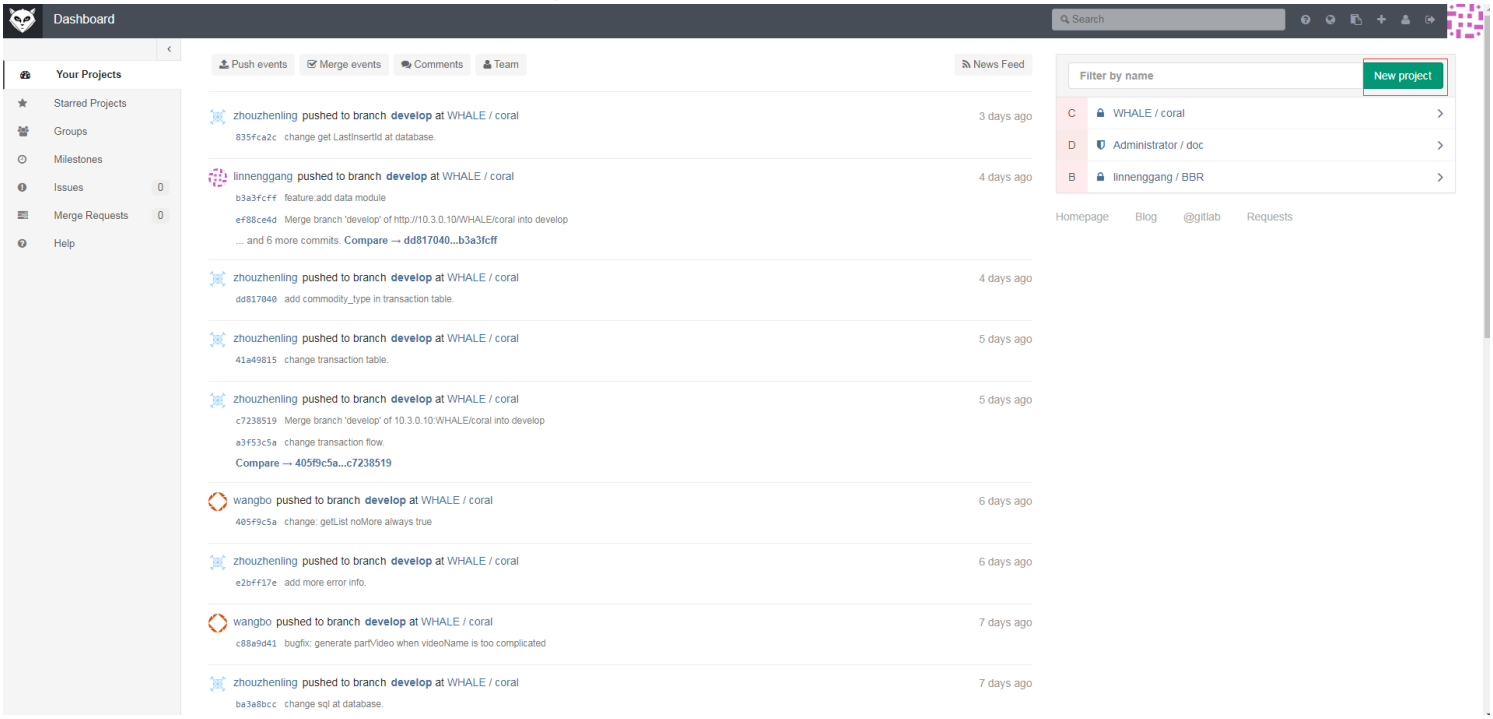
2.2 查看配置信息

要检查已有的配置信息，可以使用 git config --list 命令： ``` $ git config --list

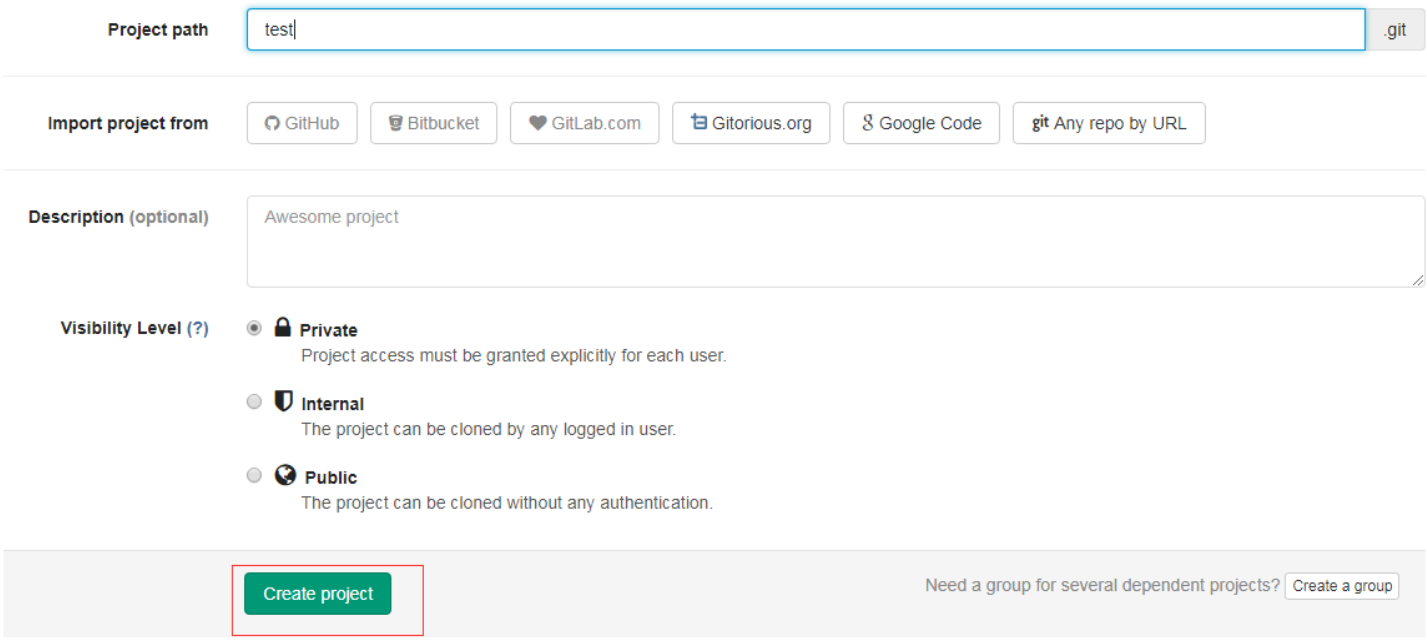
user.name=alan user.email=alan@gmail.com ```

2.3 创建远程仓库

如果是已有项目这个步骤可省。 如果是一个全新项目，则需要打开gitlab web页面， 点击新建项目如图：



进入新建项目页面， 设置项目名和项目等级， 点击create project按钮即可。创建成功后会得到一个远程仓库url， 如https://github.com/xxx/test.git



2.4 创建本地仓库

创建本地仓库有两种方法， 一种通过命令行新建， 另一种直接克隆远程仓库。一个已有项目用第二种更方便。 **初始化一个新的仓库**

可以对一个已存在的文件夹用下面的命令让它置于Git的版本控制管理之下。 创建代码目录project：

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/xxx/test.git
git push -u origin master
```

其中“git remote add origin https://github.com/xxx/test.git” 是指定本地仓库的origin为指定的url， 方便push操作， 不用每次输入一长串url。 **Clone一个仓库**
`git clone https://github.com/xxx/test.git`

2.5 创建develop分支

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

2.6 从develop分支创建Feature分支进行新功能开发

```
git checkout -b feature/xxxx develop
git status   查看工作区状态， 是否有修改文件
git add some-file  把工作时的所有变化提交到暂存区，包括文件内容修改以及新建文件
git commit -m "commit message" 将本次修改提交本地分支
git push origin feature/xxxx  将本地分支的修改合入远程Feature分支
```

2.7 把Feature分支合入远程develop分支

特性分支合入远程develop有两种方法， 一个是在本地合入feature然后再push到远程develop分支； 另一个是将feature的修改push到远程feature分支， 让后在web端发起远程feature分支合入远程develop分支的merge request，管理人员codeview后就可以在web上把修改合入远程develop分支。 **本地合入feature分支**
`git checkout develop git merge --squash feature/xxxx git commit git push origin develop` git commit 不加-m， 则会进入commit_editmsg视图， 在这里可以像编辑文件一样， 使commit messages更加清晰。需要汇总feature分支的多次的提交，使develop的log看起来清晰。

2.8 把develop分支合入远程master分支

方法雷同2.7。

2.9 打tag

如果达到一个重要的阶段，并希望永远记住那个特别的提交快照，你可以使用 git tag 给它打上标签。

```
git tag -a <tagname> -m "标签信息"
```

2.10 发布后的产品Bug修复工作流程

2.10.1 获取Bug产品的软件发布版本号 2.10.2 查找Tag `$git tag -n1 -l v*` **2.10.3 基于Tag创建分支**
`$git checkout -b hotfix-v1.0.0.170718 v1.0.0.170718` **2.10.4 修复完毕后分别合并到develop分支和master分支**
`/* 合并到develop */ $git checkout develop $git merge hotfix-v1.0.0.170718 /* 提交到远程仓库develop分支 */ $git push origin develop /* 合并到m`
2.10.5 创建新的Tag `$git tag -m "Bug#002 修复某某Bug" v1.0.1.170719 /* 推送到远程仓库 */ $git push origin v1.0.1.170719` **2.10.6 删除hotfix分支**
`$git branch -d hotfix-v1.0.0.170718`

3 git其他常用操作

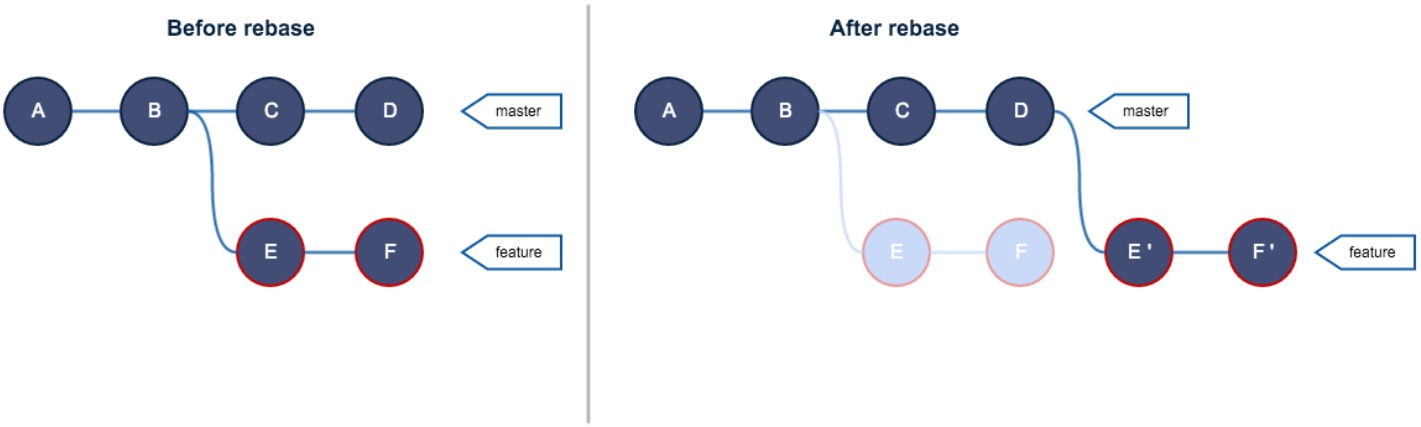
3.1 撤销操作

** 3.1.1 撤销上一次的提交，但是保留暂存区和当前修改不变** ``` git reset --soft HEAD

```
** 3.1.2 撤销上一次的提交和暂存区修改，仅保留当前修改不变**
```

$git reset --mixed HEAD `** 3.1.3` 丢弃所有修改，包括暂存区和当前目录中的修改，整体回档到上上次的提交(危险) `**` git reset --hard HEAD
`** 3.1.4` 撤销对指定文件的修改** $git checkout 文件路径 ```

3.2 git rebase



从上图可以看出，在对特征分支进行rebase之后，其等效于创建了新的提交。

3.3 git stash

暂存当前正在进行的工作， 比如想pull 最新代码， 又不想加新commit， 或者另外一种情况，为了fix 一个紧急的bug, 先stash, 使返回到自己上一个commit, 改完bug之后再stash pop, 继续原来的工作。 `$git stash $do some works $git stash pop`

3.4 冲突解决

git自动解决冲突；对比软件beyond compare

4 commit提交规范

4.1 非master分支的commit message格式

4.1.1 Commit message格式

每次提交，Commit message 都包括两个核心部分：标题 和 内容。 `<类型>: <主题> // 空一行 <内容>` 其中，标题 是必需的，内容无需过多描述的话，正文内容部分可以省略。

不管是哪一个部分，任何一行在80个字符左右。这是为了避免自动换行影响美观。

4.1.2 标题

标题部分只有一行，包括字段：类型 和 主题。

标题限制总字数在50个字符以内，以保证容易阅读。 ``` e.g. Bugfix: fixed clang version detection

1、silenced getsockopt(TCP_FASTOPEN) messages on FreeBSD. 2、re-open syslog udp socket on send error (ticket #1477).

FreeBSD returns EINVAL when getsockopt(TCP*FASTOPEN) is called on a unix domain socket, resulting in "getsockopt(TCP*FASTOPEN) ... failed" messages during binary upgrade when unix domain listen sockets are present in the configuration. Added EINVAL to the list of ignored error codes.

Previously, only unix domain sockets were reopened to tolerate cases when local syslog server was restarted. It makes sense to treat other cases (for example, local IP address changes) similarly. ``` ** 类型**

类型 用于说明 commit 的类别，只允许使用下面7个标识。 Feature：新功能（feature） Bugfix：修补bug Change：优化和改善，不会改动逻辑和具体功能等

（可选）类型后面可以加上括号，括号内填写主要变动的范围，比如按功能模块分，某模块；或按项目三层架构模式分，分数据层、控制层之类的。
`e.g. Feature(#net): 表示新增net模块 e.g. Change(#all): 表示对所有模块进行了修改` ** 主题** 主题是commit目的的简短描述，不超过50个字符, 使用祈使语气, 结尾不加句号.

4.1.3 内容

内容部分是对本次 commit 的详细描述，可以分成多行，正文在 72 个字符处换行。 使用正文解释做了什么(what)和为什么(why)，而不是如何做，以及与以前行为的对比。 于是可以这样写： ``` balabala : balabala

1、balabala balabala balabala balabala 2、balabala balabala balabala balabala

balabala balabala balabala balabala ```

4.2 master分支的commit message格式

master的commit message需要汇总dev所有commit， 参照nginx的master提交。 ``` Changes with nginx 1.15.0 05 Jun 2018

```
*) Change: the "ssl" directive is deprecated; the "ssl" parameter of the
   "listen" directive should be used instead.

*) Change: now nginx detects missing SSL certificates during
   configuration testing when using the "ssl" parameter of the "listen"
   directive.

*) Feature: now the stream module can handle multiple incoming UDP
   datagrams from a client within a single session.

*) Bugfix: it was possible to specify an incorrect response code in the
   "proxy_cache_valid" directive.

*) Bugfix: nginx could not be built by gcc 8.1.

*) Bugfix: logging to syslog stopped on local IP address changes.

*) Bugfix: nginx could not be built by clang with CUDA SDK installed;
   the bug had appeared in 1.13.8.

*) Bugfix: "getsockopt(TCP_FASTOPEN) ... failed" messages might appear
   in logs during binary upgrade when using unix domain listen sockets
   on FreeBSD.

*) Bugfix: nginx could not be built on Fedora 28 Linux.

*) Bugfix: request processing rate might exceed configured rate when
   using the "limit_req" directive.

*) Bugfix: in handling of client addresses when using unix domain listen
   sockets to work with datagrams on Linux.

*) Bugfix: in memory allocation error handling.
```