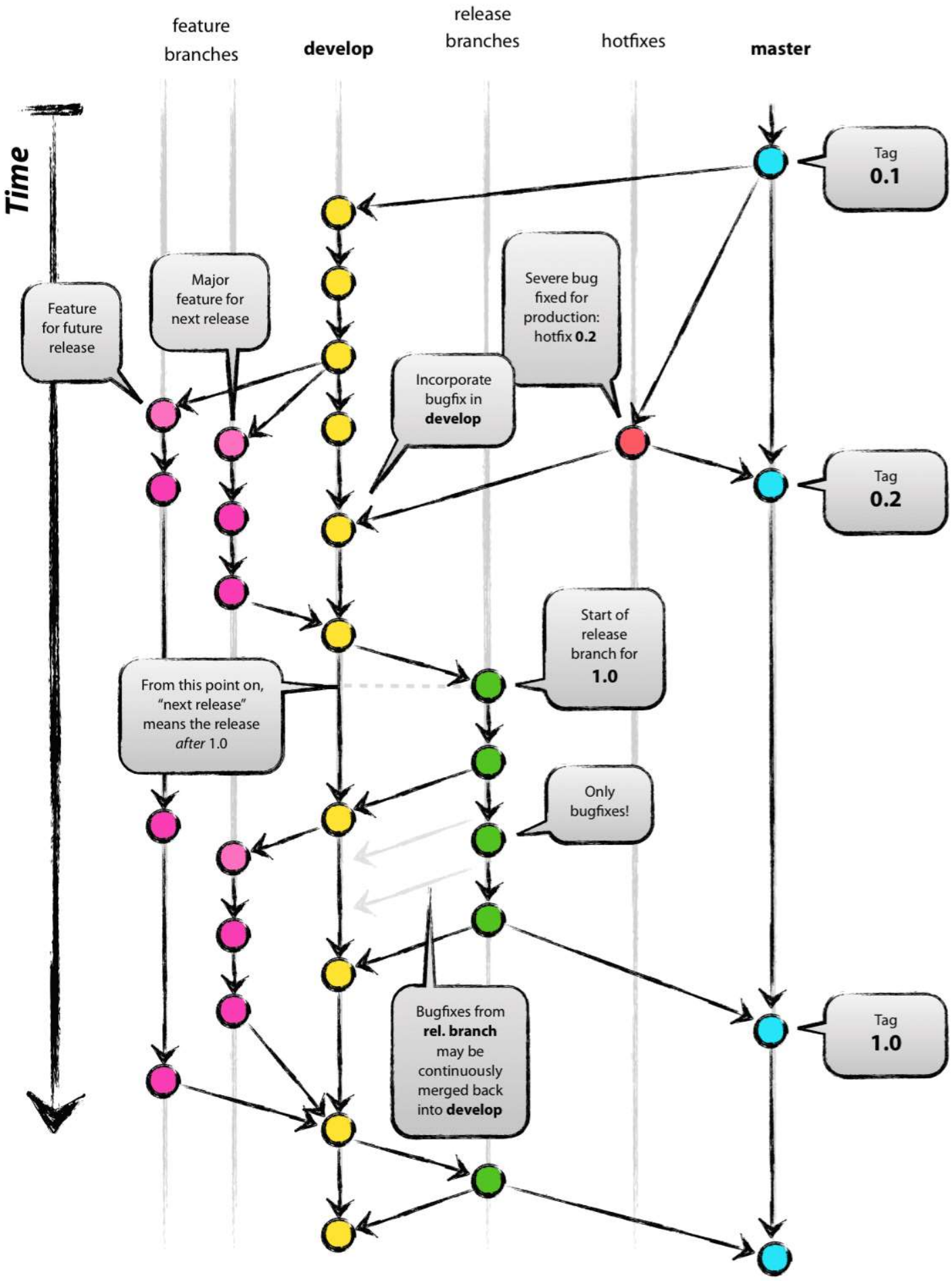
Git 分支管理及开发流程

总览


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

分支主要功能

分支名	功能
master	主分支，主要用于版本发布
develop	日常开发分支，该分支正常保存了开发的最新代码
feature	具体功能开发分支，只与 develop 分支进行交互
release	release 分支为 master 分支的未测试版本。 如某版功能全部开发完成，将 develop 分支合并到 release 分支，测试通过之后， 到了发布日期将 release 分支合并到 master 分支进行发布
hotfix	线上 Bug 修复分支

开发流程

> 主分支只有 develop 和 master 两个分支
> feature 、release 及 hotfix 为辅助分支

创建项目

1. 创建项目，新建 master 分支及 develop 分支
2. 从 develop 分支新建 feature 分支（feature 分支以功能进行命名）进行新功能开发

新功能开发

1. 从 develop 分支新建 feature 分支（feature 分支以功能进行命名）进行新功能开发
2. 新功能开发完毕，将 feature 合并到 develop 分支并删除相应的 feature 分支

版本预发布

1. 新功能开发完毕，将 feature 合并到 develop 分支并删除相应的 feature 分支
2. 当前版本所有功能开发完毕，从 develop 分支创建一个 release 分支，并交由测试进行相关测试
3. 测试提交的问题在 release 分支进行修复

版本发布

1. release 分支测试完毕，将 release 分支合并到 master 分支及 develop 分支，并删除当前 release 分支

线上Bug修复

1. 从 master 分支新建 hotfix 分支
2. 修复 Bug 之后，将 hotfix 分支合并到 master 分支及 develop 分支
3. 删除 hotfix 分支

Git commit 规范

> 本规范参考 `Angular Commit Message` 进行编写

请使用 `git commit -v` 进行提交

每个 commit message 应该包含一个 `header`、一个 `body` 和一个 `footer`，`header` 有一个特殊的格式，包含 `type(类型)`、`scope(作用域)` 和 `subject(主题)`：

```
<type>(<scope>): <subject>
<BLANK LINE>
        <body>
<BLANK LINE>
<footer>
```

`header`、`body`、`footer` 三个部分缺一不可。

注： commit message 的任何行不能超过100个字符。其中 `subject` 部分建议不超过50个字符，不能超过72个字符

```
<type>[必填]
        必须为下列选项中的一项:
                feat:     新功能
                fix:      BUG修复
                docs:     项目文档更改
                style:    代码风格修正，不影响代码功能（空格，格式化，缺少分号等）
                refactor: 重构，代码更改既不修复错误也不添加功能
                perf:     提升代码性能
                test:     添加代码测试
                chore:    构建过程或辅助工具和库（如文档生成）的更改
                ci:       脚本修改

<scope>[必填]
        用于说明 commit影响范围

<subject>[必填]
        commit 目的的简短描述,不超过50个字符

<Footer>[必填]
        放置 Redmine 任务编号
        放置写备注啥的，如果是 bug ，可以把bug id放入
        如果当前代码与上一个版本不兼容
        则 Footer 部分以BREAKING CHANGE开头

<Revert>[可选]
        如果当前 commit 用于撤销以前的 commit
        则必须以revert:开头,后面跟着被撤销 Commit 的 Header

示例:
        fix(Network): change DEFAULT_KEY

                1. change DEFAULT_KEY for ZigBee message encryption
                2. delete ...

        # 3456
        # 3460
        BREAKING CHANGE: message key has changed.
                To migrate the code follow the example below:
                ...
```

注：注意 `commit message` 格式！！！

Git 分支管理需要用到的相关命令

feature 分支

1. 从 develop 分支建一个 feature 分支，并切换到 feature 分支

```
git checkout -b myfeature develop
```

2. 合并feature 分支到 develop

```
git checkout develop

git merge --no-ff myfeature

git branch -d myfeature

git push origin develop
```

release 分支

1. 新建 release 分支

```
git checkout -b release-1.2 develop

git commit -a -m "Bumped version number to 1.2"
```

2. release 分支合并到 master 分支

```
git checkout master

git merge --no-ff release-1.2

git tag -a 1.2
```

3. release 分支合并到 develop 分支

```
git checkout develop

git merge --no-ff release-1.2
```

4. 删除 release 分支

```
git branch -d release-1.2
```

hotfix 分支

1. 新建 hotfix 分支

```
git checkout -b hotfix-1.2.1 master

git commit -a -m "Bumped version number to 1.2.1"
```

2. Fix bug

```
git commit -m "Fixed severe production problem"
```

3. buf fix 之后，hotfix 合并到 master

```
git checkout master

git merge --no-ff hotfix-1.2.1

git tag -a 1.2.1
```

4. hotfix 合并到 develop 分支

```
git checkout develop

git merge --no-ff hotfix-1.2.1
```

5. 删除 hotfix 分支

```
git branch -d hotfix-1.2.1
```

Git 常用命令集合

一、新建代码库

在当前目录新建一个Git代码库

```
$ git init
```

新建一个目录，将其初始化为Git代码库

```
$ git init [project-name]
```

下载一个项目和它的整个代码历史

```
$ git clone [url]
```

二、配置

> Git的设置文件为.gitconfig，它可以在用户主目录下（全局配置），也可以在项目目录下（项目配置）。

显示当前的Git配置

```
$ git config --list
```

编辑Git配置文件

```
$ git config -e [--global]
```

设置提交代码时的用户信息

```
$ git config [--global] user.name "[name]"
$ git config [--global] user.email "[email address]"
```

三、增加/删除文件

添加指定文件到暂存区

```
$ git add [file1] [file2] ...
```

添加指定目录到暂存区，包括子目录

```
$ git add [dir]
```

添加当前目录的所有文件到暂存区

```
$ git add .
```

添加每个变化前，都会要求确认

对于同一个文件的多处变化，可以实现分次提交

```
$ git add -p
```

删除工作区文件，并且将这次删除放入暂存区

```
$ git rm [file1] [file2] ...
```

停止追踪指定文件，但该文件会保留在工作区

```
$ git rm --cached [file]
```

改名文件，并且将这个改名放入暂存区

```
$ git mv [file-original] [file-renamed]
```

四、代码提交

提交暂存区到仓库区

```
$ git commit -m [message]
```

提交暂存区的指定文件到仓库区

```
$ git commit [file1] [file2] ... -m [message]
```

提交工作区自上次commit之后的变化，直接到仓库区

```
$ git commit -a
```

提交时显示所有diff信息

```
$ git commit -v
```

使用一次新的commit，替代上一次提交

如果代码没有任何新变化，则用来改写上一次commit的提交信息

```
$ git commit --amend -m [message]
```

重做上一次commit，并包括指定文件的新变化

```
$ git commit --amend [file1] [file2] ...
```

五、分支

列出所有本地分支

```
$ git branch
```

列出所有远程分支

```
$ git branch -r
```

列出所有本地分支和远程分支

```
$ git branch -a
```

新建一个分支，但依然停留在当前分支

```
$ git branch [branch-name]
```

新建一个分支，并切换到该分支

```
$ git checkout -b [branch]
```

新建一个分支，指向指定commit

```
$ git branch [branch] [commit]
```

新建一个分支，与指定的远程分支建立追踪关系

```
$ git branch --track [branch] [remote-branch]
```

切换到指定分支，并更新工作区

```
$ git checkout [branch-name]
```

切换到上一个分支

```
$ git checkout -
```

建立追踪关系，在现有分支与指定的远程分支之间

```
$ git branch --set-upstream [branch] [remote-branch]
```

合并指定分支到当前分支

```
$ git merge [branch]
```

选择一个commit，合并进当前分支

```
$ git cherry-pick [commit]
```

删除分支

```
$ git branch -d [branch-name]
```

删除远程分支

```
$ git push origin --delete [branch-name]
$ git branch -dr [remote/branch]
```

六、标签

列出所有tag

```
$ git tag
```

新建一个tag在当前commit

```
$ git tag [tag]
```

新建一个tag在指定commit

```
$ git tag [tag] [commit]
```

删除本地tag

```
$ git tag -d [tag]
```

删除远程tag

```
$ git push origin :refs/tags/[tagName]
```

查看tag信息

```
$ git show [tag]
```

提交指定tag

```
$ git push [remote] [tag]
```

提交所有tag

```
$ git push [remote] --tags
```

新建一个分支，指向某个tag

```
$ git checkout -b [branch] [tag]
```

七、查看信息

显示有变更的文件

```
$ git status
```

显示当前分支的版本历史

```
$ git log
```

显示commit历史，以及每次commit发生变更的文件

```
$ git log --stat
```

搜索提交历史，根据关键词

```
$ git log -S [keyword]
```

显示某个commit之后的所有变动，每个commit占据一行

```
$ git log [tag] HEAD --pretty=format:%s
```

显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

```
$ git log [tag] HEAD --grep feature
```

显示某个文件的版本历史，包括文件改名

```
$ git log --follow [file]
$ git whatchanged [file]
```

显示指定文件相关的每一次diff

```
$ git log -p [file]
```

显示过去5次提交

```
$ git log -5 --pretty --oneline
```

显示所有提交过的用户，按提交次数排序

```
$ git shortlog -sn
```

显示指定文件是什么人在什么时间修改过

```
$ git blame [file]
```

显示暂存区和工作区的差异

```
$ git diff
```

显示暂存区和上一个commit的差异

```
$ git diff --cached [file]
```

显示工作区与当前分支最新commit之间的差异

```
$ git diff HEAD
```

显示两次提交之间的差异

```
$ git diff [first-branch]...[second-branch]
```

显示今天你写了多少行代码

```
$ git diff --shortstat "@{0 day ago}"
```

显示某次提交的元数据和内容变化

```
$ git show [commit]
```

显示某次提交发生变化的文件

```
$ git show --name-only [commit]
```

显示某次提交时，某个文件的内容

```
$ git show [commit]:[filename]
```

显示当前分支的最近几次提交

```
$ git reflog
```

八、远程同步

下载远程仓库的所有变动

```
$ git fetch [remote]
```

显示所有远程仓库

```
$ git remote -v
```

显示某个远程仓库的信息

```
$ git remote show [remote]
```

增加一个新的远程仓库，并命名

```
$ git remote add [shortname] [url]
```

取回远程仓库的变化，并与本地分支合并

```
$ git pull [remote] [branch]
```

上传本地指定分支到远程仓库

```
$ git push [remote] [branch]
```

强行推送当前分支到远程仓库，即使有冲突

```
$ git push [remote] --force
```

推送所有分支到远程仓库

```
$ git push [remote] --all
```

九、撤销

恢复暂存区的指定文件到工作区

```
$ git checkout [file]
```

恢复某个commit的指定文件到暂存区和工作区

```
$ git checkout [commit] [file]
```

恢复暂存区的所有文件到工作区

```
$ git checkout .
```

重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

```
$ git reset [file]
```

重置暂存区与工作区，与上一次commit保持一致

```
$ git reset --hard
```

重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

```
$ git reset [commit]
```

重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

```
$ git reset --hard [commit]
```

重置当前HEAD为指定commit，但保持暂存区和工作区不变

```
$ git reset --keep [commit]
```

新建一个commit，用来撤销指定commit

后者的所有变化都将被前者抵消，并且应用到当前分支

```
$ git revert [commit]
```

暂时将未提交的变化移除，稍后再移入

```
$ git stash
$ git stash pop
```

十、其他

生成一个可供发布的压缩包

```
$ git archive
```