git 版本管理

总览

一个工程有一个管理员，可能有多个开发者。

有一个在服务器上的纯版本库（本文都称为 origin），此版本库常设 master 和 develop 分支，对每个开发者都设$name 和${name}-master 分支，临时分支有预发布分支 release，其他分支管理员可自行处理。

origin 上各分支状态分析。master 分支永远处于一个比较稳定并且可以发行软件的状态；develop 分支一般处于最新的开发状态；准备要发布新版本则基于 develop 创建 release 分支，此分支一般只修复 bug，这时 release 分支替代 develop 分支成为最新开发状态，确认没有问题后 release 分支 merge 回 master 和 develop 分支，并发布版本；$name 分支永远记录该开发者最新的开发状态，开发者进行日常开发和 release 开发都 push 到该分支；只有在该开发者进行了 hotfix 才会 push 到${name}-master 分支。

每个开发者都有自己的版本库，一般常设 master 和 develop 分支，其他的临时分支各个开发者自己使用。各个开发者管理 origin 上的$name 和${name}-master 分支。

日常开发者修改代码前和 push 代码前本地的 develop 分支要和 origin 上的 develop 分支同步，基于 develop 分支创建新分支修改代码，代码合并到本地的 develop 分支，并把本地的 develop 分支 push 到 origin 上的$name 分支；origin 上有 release 分支时，开发者修改代码前和 push 代码前本地的最新开发分支（可以是 develop 分支或与 origin 一致用 release 分支）要和 origin 上的 release 分支同步，基于 release 分支创建新分支修改代码，代码合并到本地的最新开发分支，并把本地的最新开发分支 push 到 origin 上的$name 分支；在版本发布后，如果出现 bug，开发者修改代码前和 push 代码前本地的 master 分支和 origin 上的 master 分支同步，基于 master 分支创建新分支修改代码，修改的代码合并到本地的 master 分支，并把本地的 master 分支 push 到 origin 上的${name}-master 分支。

管理员自己的版本库有对应 origin 上的所有分支，管理员从 origin 上拉取更新，在本地完成代码合并后再 push 回 origin。管理员管理 origin 上除各开发者的$name 和${name}-master 之外的分支。

一般不直接修改 origin 上的东西，所有的修改都使用 push 操作，所有的 push 操作都会发出邮件来通知管理员和每一个开发者。

开发者篇

日常开发一般步骤

1、克隆代码:

```
git clone ssh://gitolite@192.168.1.169/mt7620a.git ./mt7620a      克隆版本库，取名为 mt7620a
cd ./mt7620a
git checkout -b develop origin/develop                            检出 develop 分支
```

2、开始修改代码前，应先更新 develop 分支代码：

```
git checkout develop
git pull origin develop
```

3、进行代码修改，都应该创建新的分支。例如开发一个新功能，创建一个 feature-xxx 分支，它是从 develop 分

支上面分出来的：

git checkout -b feature-xxx develop　　基于 develop 分支创建一个新的分支

/* 在 feature-xxx 分支上开发 */

4、push 代码前，再次更新代码：

git checkout develop

git pull origin develop

5、代码修改完成后，合并分支代码到 develop 分支：

git checkout develop

git merge [--no-ff] feature-xxx　　将分支上的改变合并到 develop 分支

git branch -d feature-xxx　　删除 feature 分支

6、把本地的 develop 分支推送到 origin 上的$name 分支：

git push origin develop:$name

release 开发步骤

如果 origin 使用了 release 分支，开发者开始修改代码前和 push 代码前要和 origin 上的 release 分支同步，直到 release 分支结束。

1、检出 release 分支：

git fetch origin

git checkout -b release-xxx origin/release-xxx　　检出 release 分支

2、修改代码前，本地 release 分支和 origin 上的 release 分支同步代码：

git checkout release-xxx

git pull origin release-xxx　　与 origin 的 release 分支同步代码

3、创建一个新的分支来修改代码，它是从 release 分支上面分出来的：

git checkout -b fixbug-xxx release-xxx

/* 在 fixbug-xxx 分支上修改代码 */

4、push 代码前，本地的最新开发分支再次和 origin 上的最新开发分支同步代码。

4.1、一般步骤：

git checkout release-xxx

git pull origin release-xxx　　与 origin 的 release 分支同步代码

4.2、如果开始修改代码前本地 release 分支已经和 origin 上的 release 分支同步，但在 push 代码前 origin 的 release 分支已经删除了，在 push 代码前本地的 release 分支应和 origin 的 develop 分支同步：

git checkout release-xxx

git pull origin develop:release-xxx　　与 origin 的 develop 分支同步代码

5、代码修改完成后，合并分支代码到 release 分支：

git checkout release-xxx

git merge [–no-ff] fixbug-xxx　　代码合到 release 分支

git branch –d fixbug-xxx

6、本地的 release 分支 push 到 origin 上的$name 分支：

git push origin release-xxx:$name　　push 修改到 origin 上的$name 分支

7、软件发布后，删除 release 分支：

git remote prune origin　　origin 的 relase 分支结束后，本地同步 origin 已删除分支

git branch –d release-xxx　　删除本地 release 分支

hotfix 步骤

hotfix 分支是基于 master 分支分出来的，用于修复已发布版本的 bug，bug 修复后合并回 develop 和 master 分支。如果有 release 分支，就合并进 release 分支而不是 develop 分支。

1、修复 bug 前，本地的 master 分支和 origin 的 master 同步代码：

git checkout master

git pull origin master　　更新本地的 master 分支

2、创建 fotfix 分支：

git checkout -b hotfix-xxx master　　基于 master 创建一个 hotfix 分支

/* 在 hotfix-xxx 分支修复 bug */

3、push 代码前，本地的 master 分支和 origin 的 master 再次同步代码：

git checkout master

git pull origin master　　更新本地的 master 分支

4、合并代码到本地的 master 分支：

git checkout master

git merge [–no-ff] hotfix-xxx　　代码合并进 master 分支

git checkout develop　　有 release 分支则执行 git checkout release-xxx

git merge [--no-ff] hotfix-xxx　　代码合并进 develop 或 release 分支

git branch –d hotfix-xxx　　删除 hotfix 分支

5、push master 分支代码到 origin 上的${name}-master 分支：

git push origin master:${name}-master

其他事项

开发者应保持每次 commit 的 log 清楚说明进行的修改。如果日志很凌乱，且对其他开发者参考价值不大，应使用–squash 忽略每一次 commit，如下：

git checkout develop

git merge –squash feature-xxx　　如果 feature-xxx 分支的 log 很凌乱，使用–squash 来 merge

git commit –am “新的 log”　　把所有的 commit 的 log 重新写，只作为一条 log

一般 merge 推荐使用—no-ff 选项，并概述合并的内容；从 origin 更新代码使用 fetch-merge 时，如果本地要 merge 的分支没有进行过修改，一般不用加—no-ff 选项，或者直接使用 git pull；如果待合并的分支只有一次 commit，一般不用—no-ff 选项。

开发者版本库的其他分支自行处理。较好的分支管理模型参考如：
Git 分支管理策略 http://www.ruanyifeng.com/blog/2012/07/git.html
介绍一个成功的 Git 分支模型 http://www.oschina.net/translate/a-successful-git-branching-model

管理员篇

日常管理一般步骤

1、克隆代码：
git clone ssh://gitolite@192.168.1.169/mt7620a.git ./mt7620a　克隆版本库，取名为 mt7620a
cd ./mt7620a
git fetch origin　拉取 origin 上的所有更新
git branch –a　查看所有分支
git checkout -b develop origin/develop　检出 develop 分支
git checkout -b $name origin/$name　检出各个用户的分支
git checkout -b ${name}-master origin/${name}-master　检出各个用户的${name}-master 分支

2、在管理分支之前，拉取 origin 上的更新：
git fetch origin

3、管理员隔夜把各个$name 分支的更新合并到 develop 分支上，在此过程决定取舍和冲突解决：
git checkout $name　切换到某个用户的分支
git merge origin/$name　把 origin 的$name 分支的更新合并到本地$name 分支
git checkout develop　切换到 develop 分支
git merge $name　把$name 的修改 merge 到 develop 分支

4、把处理好的代码上传 origin：
git push origin develop:develop

release 管理步骤

需要发布版本时，先创建一个预发布分支，进行代码完善。预发布分支是基于 develop 分支分出来的。如果使用了 release 分支，开发者的最新版本应与 origin 的 release 分支同步，把各开发者的更新合并到 release 分支上（而不是像日常一样合并到 develop 分支）。在预发布版本结束前 develop 分支一般不动，或尽量少动。

1、创建预发布分支，xxx 为即将发布的软件的版本号：
git checkout -b release-xxx develop
git push origin release-xxx:release-xxx　把创建的 release 分支推送到 origin 上

2、在管理分支之前，拉取 origin 上的更新:
git fetch origin

3、管理员在需要时就把各个$name 分支的更新合并到 release 分支，一般频率会较高：
git checkout $name　切换到某个用户的分支

git merge origin/$name　　把 origin 的$name 分支的更新合并到本地$name 分支
git checkout release-xxx　　切换到 release 分支
git merge $name　　把$name 的修改 merge 到 release 分支

4、把处理好的代码上传 origin：
git push origin release-xxx:release-xxx

5、测试已没什么问题了，则把 release 分支的代码合并进 master 和 develop 分支，并发布软件，删除 release 分支：
git checkout master
git merge [--no-ff] release-xxx　　release 分支合并进 master
git tag –a xxx –m　　给 master 打上 tag，标志着软件正式发布了
git checkout develop
git merge [--no-ff] release-xxx　　release 分支合并进 develop
git branch -d release-xxx　　删除 release 分支

6、push 删除 release 分支的操作：
git push origin master:master　　推送 master 分支
git push origin develop:develop　　推送 develop 分支
git push origin :release-xxx　　删除 origin 上的 release 分支
git push origin xxx　　推送 tag xxx

hotfix 管理步骤

软件发布后，可能需要修补 bug。hotfix 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 develop 分支，如果有 release 分支，就合并进 release 分支而不是 develop 分支。

1、创建 hotfix 分支，xxx 一般为已发布的版本号加 1，例如已发布版本为 1.0.0，则 xxx 取 1.0.1：
git checkout -b hotfix-xxx master

2、管理分支前，拉取更新：
git fetch origin

3、合并各个${name}-master 分支的更新到 hotfix 分支：
git checkout ${name}-master　　切换到某个用户的 name-master 分支
git merge origin/${name}-master　　把 origin 的${name}-master 分支的更新合并到本地${name}-master 分支
git checkout hotfix-xxx　　切换到 hotfix 分支
git merge ${name}-master　　把${name}-master 的修改 merge 到 hotfix 分支

4、确认 bug 已修复后，合并代码到 master，如果有 release 分支则合并到 release 分支，没有 release 分支则合并到 develop 分支：
git checkout master
git merge hotfix-xxx　　合并代码到 master 分支
git tag –a xxx –m　　打上 tag
git checkout develop　　有 release 分支则执行 git checkout release-xxx
git merge hotfix-xxx　　合并代码到 develop 或 release 分支
git branch –d hotfix-xxx　　删除 hotfix 分支

5、推送更新：

git push origin master:master　　推送 master 分支

git push origin develop:develop　有 release 分支则或执行 git push origin release-xxx:release-xxx

git push origin xxx　　推送 tag xxx

其他事项

一般发布版本或者有重大更新，管理员才把 develop 的代码合并到 master 分支，而且一般需要经过 release 分支的完善。

为了让日志信息简洁些，管理员 merge 时一般不使用—no-ff 选项；但 release 分支的合并到 master 和 develop 分支时一般要加上—no-ff 选项；如果待合并的分支只有一次 commit，一般不用—no-ff 选项。