**Git源代码管理规范**

# 分支管理

使用git进行源代码管理，一般将某个项目的所有分支分为以下几条主线：

### **Master**

顾名思义，既然名字叫Master，那么该分支就是主分支的意思。master分支永远是production-ready的状态，即稳定可产品化发布的状态。

### **Develop**

这个分支就是我们平常开发的一个主要分支了，不管是要做新的feature还是需要做bug fix，都是从这个分支分出来做。在这个分支下主要负责记录开发状态下相对稳定的版本，即完成了某个feature或者修复了某个bug后的开发稳定版本。

### **Feature branches**

这是由许多分别负责不同feature开发的分支组成的一个分支系列。new feature主要就在这个分支系列下进行开发。当功能点开发测试完毕之后，就会合并到develop分支去。

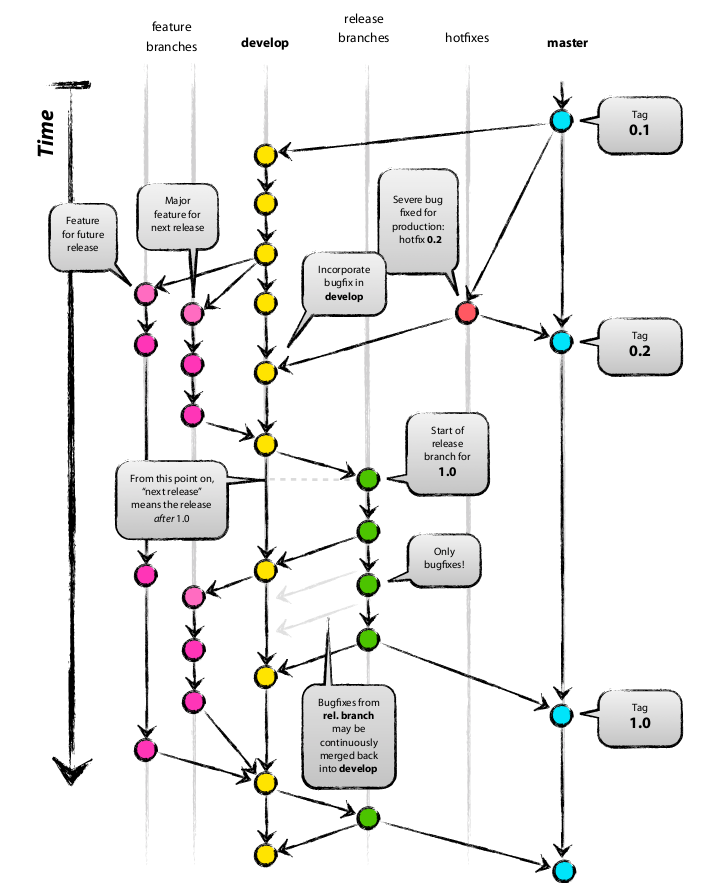
### **release branches**

这个分支系列从develop分支出来，也就是预发分支。在预发状态下，我们往往会进行预发环境下的测试，如果出现缺陷，那么就在该release分支下进行修复，修复完毕测试通过后，即分别并入master分支后develop分支，随后master分支做正常发布。

### **Hotfix branches**

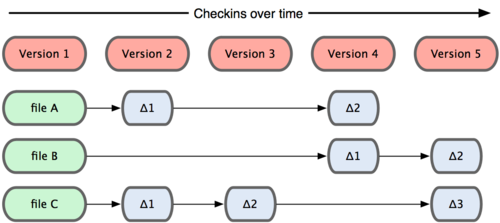
这个分支系列也就是我们常说的紧急线上修复，当线上出现bug且特别紧急的时候，就可以从master拉出分支到这里进行修复，修复完成后分别并入master和develop分支。

下面这张图将完整展示这一个流程

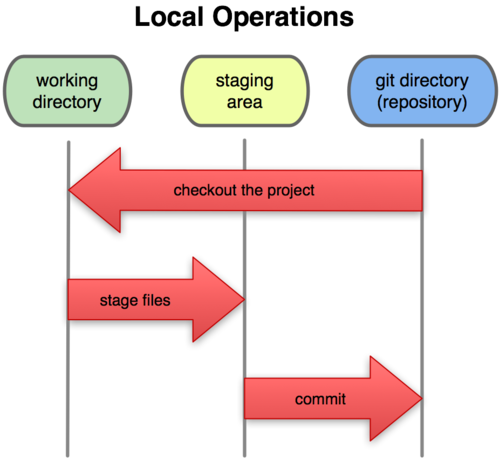


# 工作原理

Git的工作方式：



也就是说，每次提交版本变动的时候，git会保存一个快照(snapshot)。如果文件没有被更改，git也不会再次保存，而是提供一个到原来文件的链接。这样一来，git更像是一个小型的文件系统。此外，**git的所有操作都可以是本地的，仅仅在将新版本的内容上传到服务器上时才需要连接网络。**



**Git目录**（repository）是Git保存元数据和对象数据库的地方。这也是Git最重要的部分。

**工作目录**（working directory）是项目某个版本的内容。

**暂存区**（staging area）是一个简单的文件，通常包含在Git目录中。其中存储了将要进入下一次提交的信息。

Git的基本工作流程如下：



1.在工作目录中修改文件。

2.标识(stage)文件，并将文件快照添加到暂存区。

3.执行commit，将获取暂存区中的文件，并将快照永久保存到Git目录中。

# 常用命令

### **创建工程**

>> git init

### **提交修改**

>> git add 后就从修改变为暂存

>> git commit 后就从暂存变为提交。

### **提交规范**

在commit时,如果有对应PR（需求项），请在第一行写上PR号，然后再描述信息（另起行），并把涉及到改动的文件名附上。

### ****回溯****

**改错了，不过还没有git add**

>> git reset --hard

**改错了，已经git add**

>> git reset -q [files]

(其实就是 git add 的反向操作)

**改错了，已经git commit**

>> git reset --soft HEAD^

(其实就是 git commit 的反向操作)

**已经git commit，忘记写注释（PR）或者漏提交了部分文件**

如果添加注释可以直接执行命令 git commit --amend，填写注释保存

如果添加文件先执行 git add 后执行 git commit --amend

### **创建分支**

**查看分支**

>> git branch

**切换分支**

>> git checkout [branch name]

**创建分支(在当前代码的基础上)**

>> git branch [branch name]

### **合并分支**

先检出目标分支再把其他分支合并进去

>> git checkout [branch name]

>> git merge [other\_branch]

### **删除分支**

>> git branch -d [branch name]

(不能删？用这个！)

>> git branch -D [branch name]

### **标签管理**

>>git tag v1.0

### **远程操作**

克隆远程库

>> git clone

定义远程库

>> git remote

从远程库取回更新

>> git fetch

从远程库取回更新并合并

>> git pull

推送至远程库

>> git push

# 操作流程(本地)

### **准备工作**

初始化目录

>> git init

>> git add readme.md

>> git commit -m 'master init'

然后从master分支中拉出develop分支

>> git checkout -b develop

### **功能点开发**

有新的需求或功能点需要开发时， 从最新develop分支中拉出一个feature分支

>> git checkout -b [feature name]

完成feature开发后需要对feature分支进行合并操作

>> git checkout develop

>> git merge [feature name]

### **处理冲突**

当合并分支出现冲突时，需要手动将文件冲突的部分进行修改。对修改后的文件保存并重新提交。

### **产品发布**

当develop分支已经达到了一个可以发布的状态，将最新的develop分支拉出来成为一个release分支

>> git checkout -b release

假设需要一些环境配置，新建配置文件并提交

>> git add release.config

>> git commit -m 'release1'

当遇到一些预发环境下的bug，这个时候我就直接在release分支下进行修复演进，如果bug问题很大，则需要重新并入develop中，拉出新的feature进行开发重构。

如果预发一切正常，需要将release分支同时并入master分支和develop分支，master分支供线上发布，develop分支供下次开发演进。

>> git checkout master

>> git merge [release name]

>> git checkout develop

>> git merge [release name]

### **线上bug热修复**

当碰到一些线上意想不到的bug，需要紧急修复时，就直接从master分支拉出hotfixes分支进行修复。

>> git checkout master

>> git checkout -b [hotfix name]

bug修复完毕，测试通过后我们将分支合并到master和develop中去。

>> git checkout develop

>> git merge [hotfix name]

>> git checkout master

>> git merge [hotfix name]

# 远程操作

远程操作的5个常用命令

* git clone
* git remote
* git fetch
* git pull
* git push



### **从远程主机克隆一个版本库**

>> git clone <版本库的网址>

该命令会在本地主机生成一个目录，与远程主机的版本库同名。

### **管理主机名**

为了便于管理，Git要求每个远程主机都必须指定一个主机名。

不带选项的时候，git remote命令列出所有远程主机。

### **将更新取回本地**

>> git fetch <远程主机名>

默认情况下，git fetch取回所有分支（branch）的更新。如果只想取回特定分支的更新，可以指定分支名。

>> git fetch <远程主机名> <分支名>

git branch命令的-r选项，可以用来查看远程分支，-a选项查看所有分支。

取回远程主机的更新以后，可以在它的基础上，使用git checkout命令创建一个新的分支。

>> git checkout -b newBrach origin/master

也可以使用git merge命令或者git rebase命令，在本地分支上合并远程分支。

>> git merge origin/master

或者

>> git rebase origin/master

### **取回更新同时合并到本地**

git pull命令的作用是，取回远程主机某个分支的更新，再与本地的指定分支合并。

>> git pull <远程主机名> <远程分支名>:<本地分支名>

如果远程分支是与当前分支合并，则冒号后面的部分可以省略。

>> git pull origin next

上面命令表示，取回origin/next分支，再与当前分支合并。实质上，这等同于先做git fetch，再做git merge。

>> git fetch origin

>> git merge origin/next