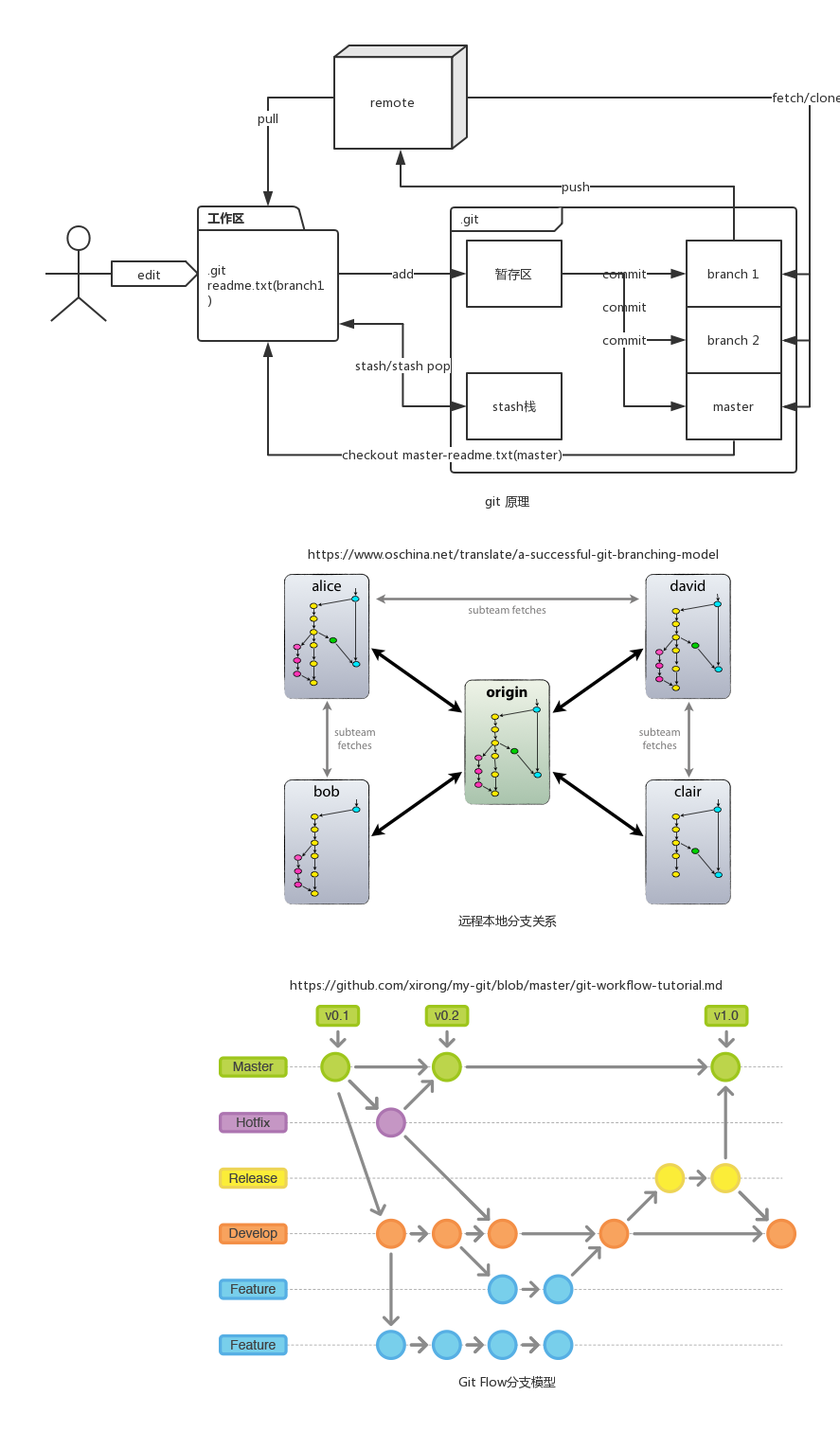
Git学习

先上图



# Git原理

首先对比svn的仓库和分支关系，在svn中分支包含在仓库中，是仓库的一部分，比如kolibri仓库包括1121分支、1205分支、baseline分支和trunk分支，每个分支都不是仓库的全部。在git中分支更像是整个仓库的某个拷贝（实际不是），代表着仓库在某个方向上的修改，也就是仓库在多个平行世界（另一个自己？hah）中的修改时间轴。在合适的时机，这些平行的修改时间轴会进行合并，通常把主线叫master，最终所有修改线都会合并到master上。

然后搞清楚几个概念：

**远程仓库**——与本地仓库存在关联的仓库，区别是url不同，任何非本地仓库的其他有关联的仓库都可以叫远程仓库。习惯上会指定其中一个远程仓库作为中心仓库，作用和svn的服务器差不多，影响力却差多了。

**工作区**——修改发生的目录区域，也可以说是仓库创建或clone下来的目录区域。一个仓库对应一个工作区（注意不是一个分支一个工作区，所以切换分支时需要确保工作区的修改被提交或被放入stash栈）。

**暂存区**——工作区的修改add后会先到暂存区等待提交，和工作区一样，一个仓库对应一个暂存区，切换分支时需要保证暂存区是干净的（空的）。

**Stash栈**——我自己虚构的一个概念，保存工作区的快照，方便进行分支切换。Stash栈的内容不能直接到暂存区或分支。

**分支**——分支是整个仓库在不同修改方向上的修改记录，并总是指向最新的修改。所有分支直接或间接的衍生于最初的分支（master），将所有分支的所有修改（包括远程仓库上的）提交描点连线能够得到一个有向的拓扑图（回滚的点线抹除）。而分支合并就是修改合并，两条线上的修改进行合并。

**标签**——commit命令生成的commitid的别名。

**关联**——不同两个仓库之间需要关联起来，这样进行fetch/pull/push等操作不需要指定在命令行指定远程的分支

从图‘远程本地分支关系’可以看出，origin作为中心仓库，所有开发从这clone仓库到本地后沿着不同的线路进行开发，开发完成后分别合并到中心仓库（也可以和其他开发直接进行合并，因为都是来自同一个仓库是有关联可合并的），合并完成后就是图中的样子。最后本地开发再同步，将其他人的修改从中心仓库中fetch下来。

# Git命令

仓库：

**git init**

—— 在当前目录创建仓库，全新仓库，未与任何其他（远程）仓库关联；

**git clone**

—— 从远程仓库拷贝一份仓库到本地当前目录，并将本地和远程仓库建立关联；

git clone url

**git remote add**

—— 将当前所在的本地仓库与远程仓库关联起来；

git remote add [name] [url] url远程仓库地址，name远程仓库名称

git remote add origin [git@server-name:path/repo-name.git](mailto:git@server-name:path/repo-name.git)

**git remote –v**

—— 查看远程仓库，也可以不需要-v参数

**git status**

—— 当前仓库状态，有哪些修改未add，哪些未commit

**git pull**

—— 同步远程仓库更新到本地仓库和工作区，相当于和远程分支进行合并。

相当于git fetch + git merge

**git fetch**

—— 拉取远程分支到本地，但不会进行合并。

比如git fetch远程库的origin/master 分支，则会拉取到本地名为origin/master的分支（不是master分支），需要在master分支上执行git merge origin/master 才能真正合并达到git pull的效果。

文件：

**git add**

—— 添加修改至暂存区

git add [file] [file] [file] … 可以一次性添加多个文件修改，或者使用git add . 添加所有文件修改

git add file2.txt file3.txt git add .

**git commit**

—— 将暂存区的内容提交到仓库，注意git的提交概念和svn是有很大差别的，svn的提交是指定修改集合的提交，比如file.txt、file1.txt一起提交，不同文件有不同的提交记录并且可以单独回滚。而git 提交后的commitid代表是当前整个分支快照。

git commit [-m msg] 记得提交注释

**git diff**

—— 比较工作区中的文件和暂存区或仓库中的文件，注意会先与暂存区中的未提交的进行比较。若暂存区中没有才会与仓库中的版本进行比较，也就是和最近一次add的版本或者commit的版本进行比较。

git diff file

**git log**

—— 查看提交日志

**git reflog**

—— 查看使用git命令历史

**git reset**

—— 回滚到指定版本

git reset –hard HEAD^ git reset –hard HEAD^^^ git reset –hard HEAD~100 三个命令分别表示恢复到当前版本的前一个版本、回到当前版本的前三个版本、回到当前版本的前100个版本。git reset –hard commitId 回到指定commitId（每次提交会生成一个id，git log可以查看，很长，可以只输入前缀，git会自定匹配）

git reset commitid filename 回滚指定文件版本到工作区（不是仓库分支中），注意需要重新commit才能生效

**git checkout -- <filename>**

—— 撤销当前工作区的所有修改，恢复到上一次add或commit的状态（和diff类似）

git checkout -- readme.txt 撤销所有readme.txt在工作区中的修改

**git rm**

—— 删除工作区中的文件，需要再通过git commit 才能提交本次删除操作到仓库

git rm readme.txt

分支：

**git branch <name>**

—— 创建新分支git branch dev 创建dev分支

git branch 不加分支名是查看所有本地分支；

git branch –r 查看远程仓库所有分支

git branch –a 查看本地和远程仓库的所有分支

git branch –d dev 删除dev分支，分支必须是已经合并到原分支（从哪个分支拉出来的）

git branch –D dev 强制删除dev分支，即使没有合并

**git checkout <name>**

—— 切换分支 git checkout dev 切换到dev分支

**git checkout –b**

—— 创建分支并切换到新创建的分支

git checkout –b dev 创建dev分支并切换到dev分支

**git merge --no-ff -m “msg” <name>**

—— 合并指定分支到当前分支，请注意--no-ff参数，表示禁用Fast forward（通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息，就看不出来曾经做过合并。所以合并时一定要加上—no-ff），还有提交注释不能忘。Fast forward模式就是自从拉分支后，到合并时原分支没有任何的更新提交，则直接修改原分支的head指针即可。

当前处于master 分支 git merge dev 命令相当于合并dev到master

**git log --graph**

—— 查看分支合并图，命令下这个图挺有意思

远程分支

**git push [alias] [branch]**

—— 推送本地分支修改到远程仓库对应分支，这里的对应是指关联起来的分支

git push origin dev 将本地dev分支更新推送到远程origin仓库的同名分支下，git push 没有参数情况是推送当前分支更新到远程同名分支中

**git branch --set-upstream dev origin/dev**

—— 分支关联，也就是在第一次push之后，需要设置本地和远程对应分支的关联，否则无法直接进行pull操作。

**git push -u [alias] [branch]**

—— 推送本地分支到远程仓库并关联起来，注意若没有-u参数，本地分支和远程分支没有关联，无法直接pull下来。该 命令相当于**git push [alias] [branch]**和**git branch --set-upstream dev origin/dev**两个命令

**git checkout -b dev origin/dev**

—— 拉取远程仓库分支，即当前本地没有对应的分支，使用该命令从远程仓库拉取新分支下来。上面命令表示拉取远程仓库的dev分支到本地，并命名为dev分支。（默认clone下来的仓库只有master分支，其他分支需要通过该命令拉取）

标签：

**git tag <name> [commitId]**

—— 给当前分支打标签，也就是给当前分支的最近一次提交的commitid命名一个别名

没有参数是查看所有的标签，也可以通过指定commitId来给不是最近一次的提交打标签

git tag -a <tagname> -m "msg" 可以指定额外标签信息

git tag -d <tagname> 删除标签

**git show <tagname>**

—— 显示标签详情

**git push origin <tagname>**

—— 推送指定标签到远程仓库，git push origin --tags一次性推送全部标签

删除远程标签 先git tag –d 删除本地标签，然后git push origin :refs/tags/<tagname> 删除远程标签

工作区：

**git stash**

—— 保存当前工作区状态

git stash pop 将stash栈顶的工作区状态弹出

git stash list 列出当前stash栈的内容

git stash apply <stashid> 获取指定的stash栈内容，stashid可以通过git stash list获知

git stash drop <stashid> 删除stash栈指定内容

# 分支策略

建议使用成熟的git-flow模型，参考<https://github.com/xirong/my-git/blob/master/git-workflow-tutorial.md>

# Git 配置

Git 的仓库配置>全局配置>系统配置

git config [--local|--global|--system] [-e|-l]

[编辑|查看][仓库|全局|系统]配置

git config [--local|--global|--system] [--add|--unset]

# 参考：

<https://github.com/xirong/my-git/blob/master/git-workflow-tutorial.md>

<https://www.oschina.net/translate/a-successful-git-branching-model>

<http://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>

<http://blog.csdn.net/hutaoer06051/article/details/8275069>