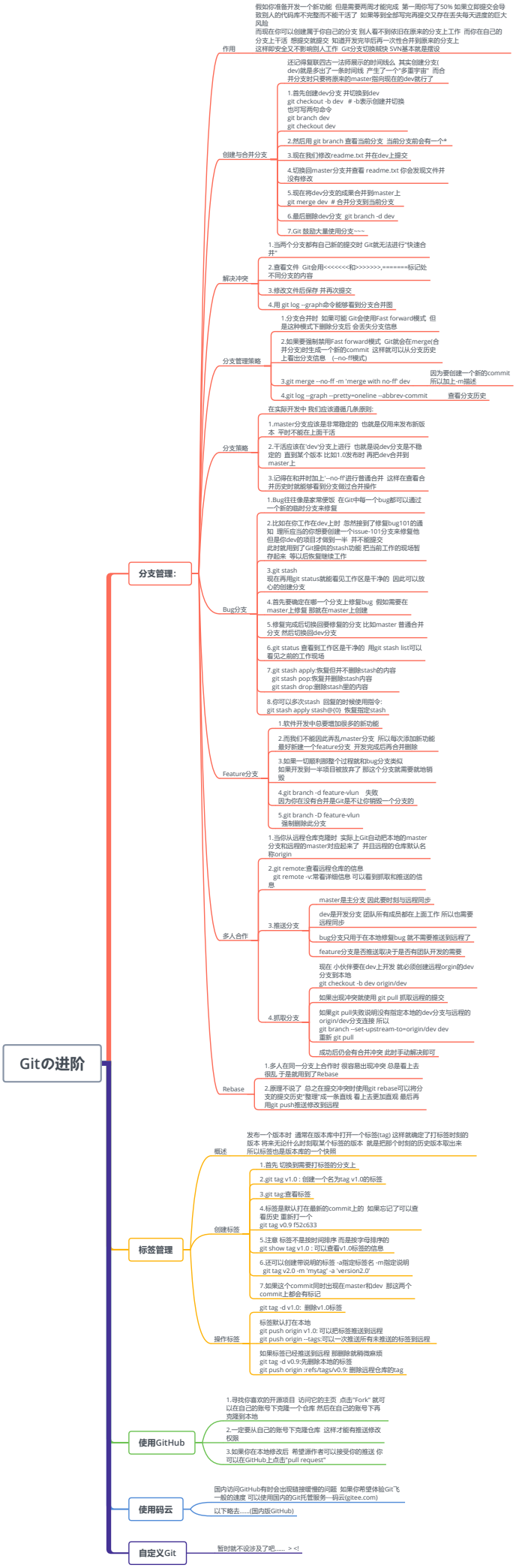
Gitの进阶
分支管理:
作用
假如你准备开发一个新功能 但是需要两周才能完成 第一周你写了50% 如果立即提交会导致别人的代码库不完整而不能干活了 如果等到全部写完再提交又存在丢失每天进度的巨大风险
而现在你可以创建属于你自己的分支 别人看不到依旧在原来的分支上工作 而你在自己的分支上干活 想提交就提交 知道开发完毕后再一次性合并到原来的分支上
这样即安全又不影响别人工作 Git分支切换赋快 SVN基本就是摆设
创建与合并分支
还记得复联四古一法师展示的时间线么 其实创建分支(dev)就是多出了一条时间线 产生了一个"多重宇宙" 而合并分支时只要将原来的master指向现在的dev就行了
1.首先创建dev分支 并切换到dev
git checkout -b dev # -b表示创建并切换
也可写两句命令
git branch dev
git checkout dev
2.然后用 git branch 查看当前分支 当前分支前会有一个*
3.现在我们修改readme.txt 并在dev上提交
4.切换回master分支并查看 readme.txt 你会发现文件并没有修改
5.现在将dev分支的成果合并到master上
git merge dev # 合并分支到当前分支
6.最后删除dev分支 git branch -d dev
7.Git 鼓励大量使用分支~~~
解决冲突
1.当两个分支都有自己新的提交时 Git就无法进行"快速合并"
2.查看文件 Git会用<<<<<<<和>>>>>>>,=======标记处不同分支的内容
3.修改文件后保存 并再次提交
4.用 git log --graph命令能够看到分支合并图
分支管理策略
1.分支合并时 如果可能 Git会使用Fast forward模式 但是这种模式下删除分支后 会丢失分支信息
2.如果要强制禁用Fast forward模式 Git就会在merge(合并分支)时生成一个新的commit 这样就可以从分支历史上看出分支信息 (--no-ff模式)
3.git merge --no-ff -m 'merge with no-ff' dev
因为要创建一个新的commit 所以加上-m描述
4.git log --graph --pretty=oneline --abbrev-commit
查看分支历史
分支策略
在实际开发中 我们应该遵循几条原则:
1.master分支应该是非常稳定的 也就是仅用来发布新版本 平时不能在上面干活
2.干活应该在'dev'分支上进行 也就是说dev分支是不稳定的 直到某个版本 比如1.0发布时 再把dev合并到master上
3.记得在和并时加上'--no-ff'进行普通合并 这样在查看合并历史时就能够看到分支做过合并操作
Bug分支
1.Bug往往像是家常便饭 在Git中每一个bug都可以通过一个新的临时分支来修复
2.比如在你工作在dev上时 忽然接到了修复bug101的通知 理所应当的你想要创建一个issue-101分支来修复他 但是你dev的项目才做到一半 并不能提交
此时就用到了Git提供的stash功能 把当前工作的现场暂存起来 等以后恢复继续工作
3.git stash
现在再用git status就能看见工作区是干净的 因此可以放心的创建分支
4.首先要确定在哪一个分支上修复bug 假如需要在master上修复 那就在master上创建
5.修复完成后切换回要修复的分支 比如master 普通合并分支 然后切换回dev分支
6.git status 查看到工作区是干净的 用git stash list可以看见之前的工作现场
7.git stash apply:恢复但并不删除stash的内容
git stash pop:恢复并删除stash内容
git stash drop:删除stash里的内容
8.你可以多次stash 回复的时候使用指令:
git stash apply stash@{0} 恢复指定stash
Feature分支
1.软件开发中总要增加很多的新功能
2.而我们不能因此弄乱master分支 所以每次添加新功能最好新建一个feature分支 开发完成后再合并删除
3.如果一切顺利那整个过程就和bug分支类似
如果开发到一半项目被放弃了 那这个分支就需要就地销毁
4.git branch -d feature-vlun 失败
因为你在没有合并是Git是不让你销毁一个分支的
5.git branch -D feature-vlun
强制删除此分支
多人合作
1.当你从远程仓库克隆时 实际上Git自动把本地的master分支和远程的master对应起来了 并且远程的仓库默认名称origin
2.git remote:查看远程仓库的信息
git remote -v:常看详细信息 可以看到抓取和推送的信息
3.推送分支
master是主分支 因此要时刻与远程同步
dev是开发分支 团队所有成员都在上面工作 所以也需要远程同步
bug分支只用于在本地修复bug 就不需要推送到远程了
feature分支是否推送取决于是否有团队开发的需要
4.抓取分支
现在 小伙伴要在dev上开发 就必须创建远程orgin的dev分支到本地
git checkout -b dev origin/dev
如果出现冲突就使用 git pull 抓取远程的提交
如果git pull失败说明没有指定本地的dev分支与远程的origin/dev分支连接 所以
git branch --set-upstream-to=origin/dev dev
重新 git pull
成功后仍会有合并冲突 此时手动解决即可
Rebase
1.多人在同一分支上合作时 很容易出现冲突 总是看上去很乱 于是就用到了Rebase
2.原理不说了 总之在提交冲突时使用git rebase可以将分支的提交历史"整理"成一条直线 看上去更加直观 最后再用git push推送修改到远程
标签管理
概述
发布一个版本时 通常在版本库中打开一个标签(tag) 这样就确定了打标签时刻的版本 将来无论什么时刻取某个标签的版本 就是把那个时刻的历史版本取出来
所以标签也是版本库的一个快照
创建标签
1.首先 切换到需要打标签的分支上
2.git tag v1.0 : 创建一个名为tag v1.0的标签
3.git tag:查看标签
4.标签是默认打在最新的commit上的 如果忘记了可以查看历史 重新打一个
git tag v0.9 f52c633
5.注意 标签不是按时间排序 而是按字母排序的
git show tag v1.0 : 可以查看v1.0标签的信息
6.还可以创建带说明的标签 -a指定标签名 -m指定说明
git tag v2.0 -m 'mytag' -a 'version2.0'
7.如果这个commit同时出现在master和dev 那这两个commit上都会有标记
操作标签
git tag -d v1.0: 删除v1.0标签
标签默认打在本地
git push origin v1.0: 可以把标签推送到远程
git push origin --tags:可以一次推送所有未推送的标签到远程
如果标签已经推送到远程 那删除就稍微麻烦
git tag -d v0.9:先删除本地的标签
git push origin :refs/tags/v0.9: 删除远程仓库的tag
使用GitHub
1.寻找你喜欢的开源项目 访问它的主页 点击"Fork" 就可以在自己的账号下克隆一个仓库 然后在自己的账号下再克隆到本地
2.一定要从自己的账号下克隆仓库 这样才能有推送修改权限
3.如果你在本地修改后 希望源作者可以接受你的推送 你可以在GitHub上点击"pull request"
使用码云
国内访问GitHub有时会出现链接缓慢的问题 如果你希望体验Git飞一般的速度 可以使用国内的Git托管服务---码云(gitee.com)
以下略去.....(国内版GitHub)
自定义Git
暂时就不说涉及了吧..... > <!