团队 git 协同开发那些事儿

前言

回想当初我在大学自学了编程技术，然后找工作很多公司给了我offer，最后选择了一家我自认为不错的，就是我现在工作的这家外企。

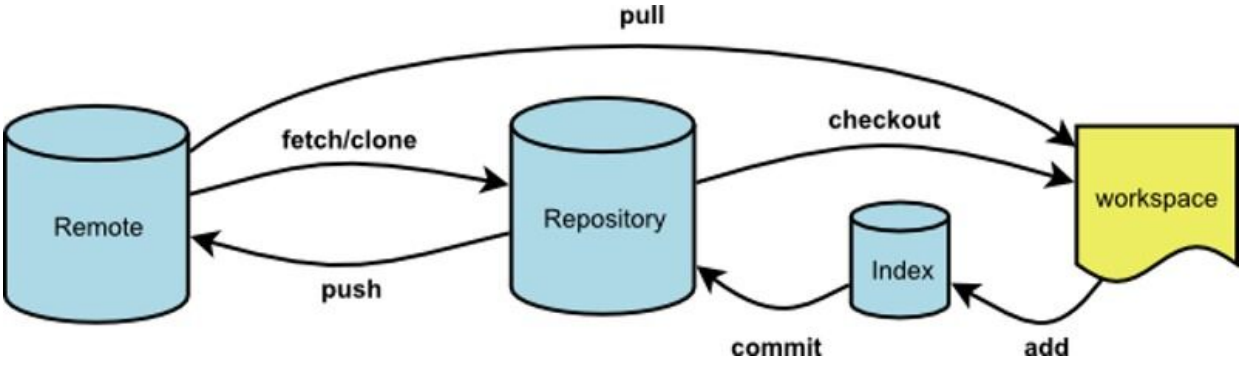
作为新人的我进了公司也和很多刚刚毕业的同学一样根本不知道公司的团队开发是什么样，流程是怎么样进行的，用什么工具提高开发效率，团队写的代码怎么共享。这些想必对于一个刚刚步入社会的新人都是一样的感受。这些是我们真正的开始有团队意识的一个开始，也是我们团队合作的开始,所以这是我们必须过的一关。

然后我明白了：**既然改变不了风的方向，那么我们只能是改变自己帆的航向。**

进入工作的第一天我就被告知需要会使用Git版本工具。于是我就开始了学习和使用Git之路，因为这是公司团队合作的基本工作技能之一。由于我在前面自己学习编程已经学会了自己快速分析问题，然后解决问题。当我在面对了这个新的陌生的技术的时候，我也学会分析和解决我当时遇到的新问题。这是我想传递的最重要的信息，无论何时我们都会遇到新的问题，关键是我们要学会分析问题和解决问题，这样的能力培养非常重要。

当时我在谷歌上查了下，**Git** 是一个开源的分布式版本控制系统，用以有效，高速的处理很小到非常大的项目版本管理。然后又介绍了很多内容，但是当我看到项目管理的时候我就已经明白其用处了。

很多时候我们学技术很重要的一点就是实践和用，这是最简单也是最真实的。我当时就是想到了技术的实用性，所以我对Git有了个整体的认识之后我就开始了我使用Git之路。可能很多人都可以在网上查到关于Git的文章和信息，但是最重要的一点是我们要快速的学会用，这个很关键！



待装好环境之后，打开命令行（这是我喜欢的方式，因为在linux下工作）首先就是跟着网上找的教程下了第一个Git的命令，但是我觉得其实并不是很使用，因为很多的时候人

的开发水平和理解水平都是不一样的，环境可能也不一样，所以别人写的文章并不是最适合自己的。那么怎么才可以快速的开发和使用呢？

我当时在想了这个问题之后我做的是另个事情，我是从Git的命令上开始学习和使用的，在用Git的命令中去理解，这对于一个新人来说是最好不过的事情了。 很多时候，我们都是觉得自己对新的技术不会，也不知道该怎么做，即使我们了解了一些，我们仍然不敢下手去做，因为我们不知道从什么地方下手，这也是困扰我们的地方。这就是我想说的也是对于新技术学习的看法，**面对新的技术在有了基本的认识之后，实践才会让你学到更多**。所以我的建议就是尽量在短期内去使用git的常用命令，必须要弄懂每个命令的含义和实现的功能。

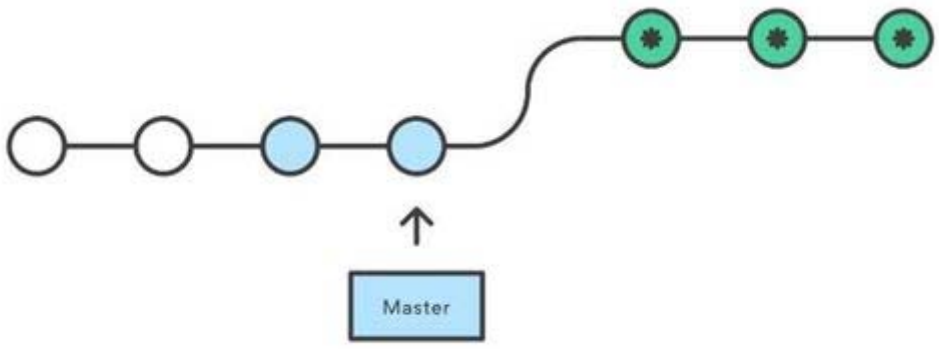
从项目开发中学习

项目开发的基本流程，在你们项目服务仓库创建一个总的项目仓库，项目开发的每个人都需要克隆一个项目仓库到你们自己个人仓库。这样你们就都有了项目仓库地址。公司一个完整的项目开发基本现在都是前后分离的，也就是开端开发和后端开发，各自分开，然后等接口写好，前端界面写好，然后进行对接。每次的修改代码和调整都会有新的变化，也会产生新的历史。这个时候你就需要提交你本地代码到远程仓库，在提交到远程的仓库的时候，自己的代码是不能提交到主分支master上的，你只能提交到dev分支上面，发起合并请求，然后由团队其他人审核过代码，才可以合并到主分支master上面，这样就完成了一次完整的提交。也是一次合乎规范的项目代码提交。

但是多人开发的时候很多人都会提交代码到主仓库，团队其他成员不可能随时都pull主仓库代码和主仓库保持一致。这样你提交的代码就会和主仓库不一致，发生冲突。相信这是我们很多人都会遇到的问题。那么有没有解决的办法呢？答案是肯定有的。

- 第一种（我建议大家使用，大家可以了解下）：强制提交忽略冲突，这是个不错的注意吧大家肯定喜欢，这个适合自己一个人玩，不适合团队合作。**git push –f origin master**。
- 第二种：就是代码哪里报错冲突了，你就解决冲突问题。

在解决了所有冲突之后，重新提交代码，然后你的代码就和主仓库保持一致了。



这个就是解决冲突后得到的仓库树状图。

在这里可能新人会有疑问，对于分支啊，合并分支啊，冲突啊等。

什么是分支

分支就是便于多人在同一项目中的协作开发。比方说：每个人开发不同的功能，在各自的分支开发过程中互不影响，完成后都提交到dev分支。极大的提高了开发的效率。

合并分支

每个人创建一个分支进行开发，当开发完成，需要合并到master分支的时候，就需要用到合并的命令。

什么是冲突

合并的时候，有可能会产生冲突。

冲突的产生是因为在合并的时候，不同分支修改了相同的位置。所以在合并的时候git不知道那个到底是你想保留的，所以就提出疑问（冲突提醒）让你自己手动选择想要保留的内容，从而解决冲突。

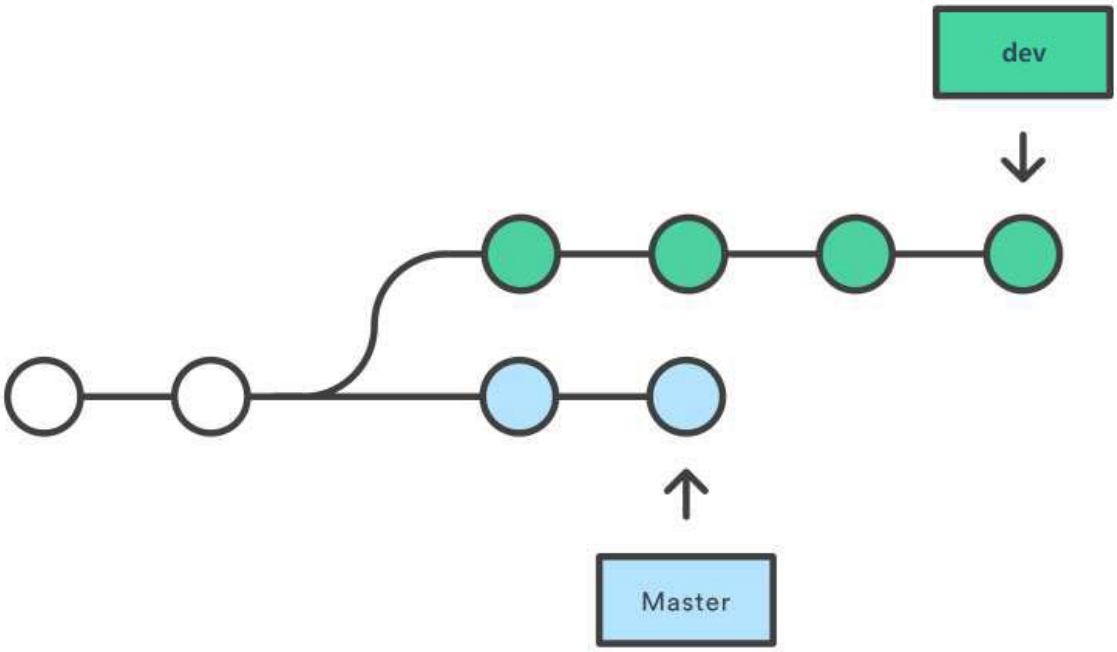
这些是最基本的操作命令，还有一个最为用处多的如merage和rebase，可能很多人都用过，但是这个在这里我详细为大家解释下这个marge和rebase吧。

git rebase 和git merge 做的事其实是一样的。它们都被设计来将一个分支的更改并入另一个分支，只不过方式有些不同。

想象一下，你刚创建了一个专门的分支开发新功能，然后团队中另一个成员在master分支上添加了新的提交。

git merge

假设我们有一个主分支 master 及一个开发分支 dev,然后合并提交(merge commit)将两个分支的历史连在了一起。



Merge好在它是一个安全的操作。现有的分支不会被更改，避免了rebase潜在的缺点（后面会说）。

另一方面，这同样意味着每次合并更改时dev会引入一个外来的合并提交。如果master非常活跃的话，这或多或少会污染你的分支历史。虽然高级的git log 选项可以减轻这个问题，但对于团队的开发者来说，还是会增加项目难度。这是git merge的一个常见问题。

git rebase

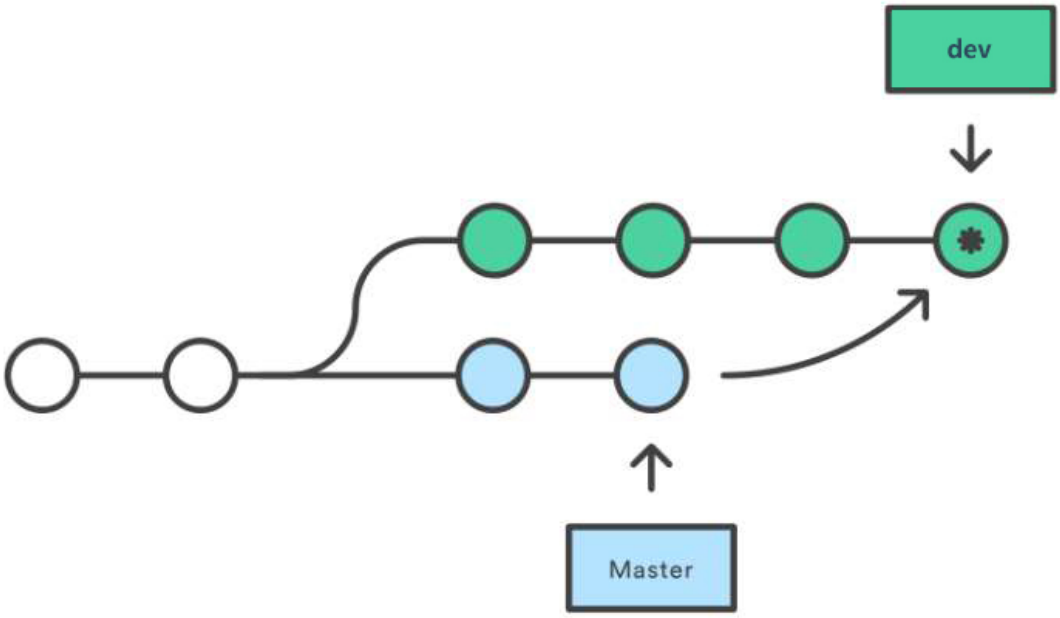
rebase最大的好处是你的项目历史会非常整洁。首先，它不像git merge 那样引入不必要的合并提交。其次， rebase导致最后的项目历史呈现出完美的。

线性——你可以从项目终点到起点浏览而不需要任何的Fork。这让你更容易使用git log 、git bisect 和gitk 来查看项目历史。

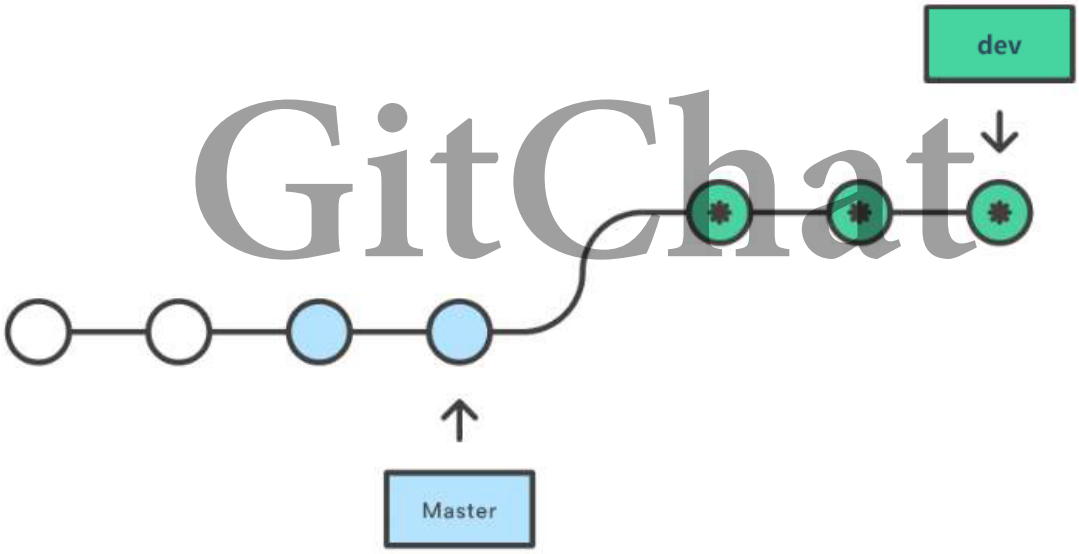
不过，这种简单的提交历史会带来两个后果：安全性和可跟踪性。如果你违反了Rebase黄金法则，重写项目历史可能会给你的协作工作流带来灾难性的影响。此外，rebase不会有合并提交中附带的信息——你看不到feature分支中并入了上游的哪些更改。

我想用两张图来介绍rebase和merge的区别：

git merge:



git rebase:



你使用rebase之前需要知道的知识点都在这了。如果你想要一个干净的、线性的提交历史，没有不必要的合并提交，你应该使用git rebase 而不是git merge 来并入其他分支上的更改。

另一方面，如果你想要保存项目完整的历史，并且避免重写公共分支上的commit， 你可以使用git merge。两种选项都很好用，但至少你现在多了git rebase这个选择。

上面介绍这些都是最基础的入门，具体的我们在分享的时候会给大家介绍。

下面是我整理出来的一些在团队开发合作的时候常用的命令，只要好好练习和使用，基本在团队开发中就可以轻松自如了。

配置帐号信息：

```
git config -e [–global] # 编辑Git配置文件

git config –global user.name keke #编辑姓名

git config –global user.email keke.li@gmail.com #编辑邮箱地址

git config –list #查看配置的信息

git help config #获取帮助信息
```

配置自动换行：

```
git config –global core.autocrlf input #提交到git是自动将换行符转换
```

配置密钥：

```
ssh-keygen -t rsa –C “emailAddress” #邮箱账户地址

ssh -T git@github.com #测试是否成功
```


配置别名，git的命令没有自动完成功能，可以配置别名简化命令名称：

```
git config –global alias.st status #git st

git config –global alias.co checkout #git co

git config –global alias.br branch #git br

git config –global alias.ci commit #git ci
```

本地新建仓库：

```
git init #初始化

git status #获取状态

git add [file1] [file2] … # .或*代表全部添加或者-A

git commit -m “message” #将内容提交,在冒号里面最好填写英文注释

git remote add origin git@github.com ##此时本地工程与远程仓库已经建立了联系

git push -u origin master #将本地的内容同步到远程仓库中
```

```
git show [十六进制码] #显示某一个特定的提交的日志

git log #查看自己提交的历史

git ls-files -u #查看冲突未处理的文件列表

git pull origin master #获取最新的远程仓库代码

git stash list #获取暂存列表
```

从现有仓库克隆：

```
git clone https://github.com/KeKe-Li/shadowsocks #把远程项目克隆到本地

git add -A #跟踪新文件

git add -u [path] # 添加[指定路径下]已跟踪文件

rm &git rm # 移除文件

git rm -f * # 移除文件

git rm –cached * # 停止追踪指定文件，但该文件会保留在工作区

git mv file_from file_to # 重命名跟踪文件

git log # 查看自己提交的历史

git commit # 提交更新

git commit [file1] [file2] … # 提交指定文件

git commit -a #跳过使用暂存区域，把所有已经跟踪过的文件暂存起来一并提交

git commit –amend #修改最后一次提交

git commit -v #提交时显示所有diff信息

git reset HEAD *#取消已经暂存的文件

git reset –soft HEAD * #重置到指定状态，不会修改索引区和工作树

git reset –hard HEAD * #重置到指定状态，会修改索引区和工作树

git reset – files #重置index区文件

git reset –hard 11d881aea55e844dc0ebf0f3e5bf12a3ca999001 # 还原指定的提交之前的状态
```

```
git revert HEAD~ #还原前前一次操作

git checkout – file #取消对文件的修改（从暂存区——覆盖worktree file）

git checkout branch | tag|commit – file_name#从仓库取出file覆盖当前分支

git checkout – .#从暂存区取出文件覆盖工作区

git diff file #查看指定文件的差异

git diff –stat #查看简单的diff结果

git diff #比较Worktree和Index之间的差异

git diff –cached #比较Index和HEAD之间的差异

git diff HEAD #比较Worktree和HEAD之间的差异

git diff branch #比较Worktree和branch之间的差异

git diff branch1 branch2 #比较两次分支之间的差异

git diff commit commit #比较两次提交之间的差异
```


git log 查看最近的提交日志：

```
git log –pretty=oneline #单行显示提交日志

git log –graph # 图形化显示

git log –abbrev-commit # 显示log id的缩写

git log -num #显示第几条log（倒数）

git log –stat # 显示commit历史，以及每次commit发生变更的文件

git log –follow [file] # 显示某个文件的版本历史，包括文件改名

git log -p [file] # 显示指定文件相关的每一次diff

git stash #将工作区现场（已跟踪文件）储藏起来，等以后恢复后继续工作。

git stash list #查看保存的工作现场

git stash apply #恢复工作现场

git stash drop #删除stash内容

git stash pop #恢复的同时直接删除stash内容
```

> git stash apply stash@{0} #恢复指定的工作现场，当你保存了不只一份工作现场时。

创建分支：

> git branch -b test #创建并切换test分支
>
> git branch #列出本地分支
>
> git branch -r #列出远端分支
>
> git branch -a #列出所有分支
>
> git branch -v #查看各个分支最后一个提交对象的信息
>
> git branch –merge #查看已经合并到当前分支的分支
>
> git branch –no-merge #查看为合并到当前分支的分支
>
> git branch branch [branch|commit|tag] # 从指定位置出新建分支
>
> git branch –track branch remote-branch # 新建一个分支，与指定的远程分支建立追踪关系
>
> git branch -m old new #重命名分支
>
> git branch -d test #删除test分支
>
> git branch -D test #强制删除test分支
>
> git branch –set-upstream dev origin/dev #将本地dev分支与远程dev分支之间建立链接
>
> git checkout test #切换到test分支
>
> git checkout -b test dev#基于dev新建test分支，并切换
>
> git merge test#将test分支合并到当前分支
>
> git merge –squash test # 合并压缩，将test上的commit压缩为一条
>
> git cherry-pick commit #拣选合并，将commit合并到当前分支
>
> git cherry-pick -n commit #拣选多个提交，合并完后可以继续拣选下一个提交‘’

Rebase:

> git rebase master #将master分之上超前的提交，变基到当前分支

> git rebase –onto master 119a6 #限制回滚范围，rebase当前分支从119a6以后的提交
>
> git rebase –interactive #交互模式
>
> git rebase –continue# 处理完冲突继续合并
>
> git rebase –skip# 跳过
>
> git rebase –abort# 取消合并

Merge

> git merge origin/branch#合并远端上指定分支

远程仓库的操作：

> git fetch origin remotebranch[:localbranch] #从远端拉去分支[到本地指定分支]
>
> git pull origin remotebranch:localbranch #拉去远端分支到本地分支
>
> git push origin branch #将当前分支，推送到远端上指定分支
>
> git push origin remote branch # 删除远端指定分支
>
> git push origin remote branch –delete # 删除远程分支
>
> git branch -dr branch # 删除本地和远程分支
>
> git checkout -b [–track] test origin/dev#基于远端dev分支，新建本地test分支[同时设置跟踪]

git是一个分布式代码管理工具，所以可以支持多个仓库，服务器上的仓库在本地称之为remote.

> git remote add origin #地址
>
> git remote #显示全部远程仓库
>
> git remote –v #显示全部源+详细信息
>
> git remote rename origin1 origin2 #重命名
>
> git remote rm origin #删除
>
> git remote show origin #查看指定源的全部信息

标签：

> git tag #列出现有标签
>
> git tag -a v0.1 -m ‘my version’ #新建带注释标签
>
> git checkout tagname #切换到标签
>
> git push origin v1.5 #推送分支到远程仓库上
>
> git push origin –tags #一次性推送所有分支
>
> git tag -d v0.1 #删除标签
>
> git push origin :refs/tags/v0.1 #删除远程标签
>
> git push origin #推送标签到远程仓库
>
> git push origin :refs/tags/ #删除远程标签需要先删除本地标签
>
> git checkout # 放弃工作区的修改
>
> git reflog #显示本地执行过git命令
>
> git remote set-url origin #修改远程仓库的url
>
> git whatchanged –since=’2 weeks ago’ #查看两个星期内的改动
>
> git ls-files –others -i –exclude-standard #展示所有忽略的文件
>
> git clean -X -f #清除gitignore文件中记录的文件
>
> git status –ignored #展示忽略的文件

强制推送远程仓库：

> git push -f #强制推送
>
> git log –all –grep=’’ #在commit log中查找相关内容

以上这些命名是我在团队开发接触到的，相信只要你也熟练运行，你们的团队开发效率将会成倍的提高！