git init : 初始化仓库

git add readme.txt : 添加文件到缓冲区

git commit -m ‘第一次添加文件’：提交文件到本地仓库

git status: 查看本地仓库状态，查看是否有文件在缓冲区或者未提交到本地仓库

git log : 查看每次提交的版本号以及相应的注释

git reflog ： 查看使用过的历史命令

git reset --hard 48bb830afe30514a47f34ee7ccc4ac06a4e7860f : 回退到指定版本

回退版本：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；

git checkout -- readme.tx : 回退到修改之前的版本

一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

git reset HEAD readme.txt : 将缓冲区的修改回退到工作区的修改

git rm 123.txt : 删除本地文件已经版本库中 的文件

二、远程库

git remote add origin [git@github.com:muyan001/relearngit.git](mailto:git@github.com:muyan001/relearngit.git) ：添加一个远程仓库名字，并和本地库进行关联

git push -u origin master ：第一次推送master分支的所有内容；

git clone [git@github.com:muyan001/cloneRepository.git](mailto:git@github.com:muyan001/cloneRepository.git) ：按照远程库克隆到本地库

三、关于分支

3.1 创建和合并分支

git branch dev：创建一个本地分支

git branch：查看本地所有分支，当前分支前面会标一个\*号。

git checkout dev：切换分支

git merge main：用于合并指定分支到当前分支

git branch -d dev：删除一个分支

git log *--graph --pretty=oneline --abbrev-commit：查看分支历史信息*

3.2 解决冲突

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

3.3 bug分支

软件开发中，bug就像家常便饭一样。有了bug就需要修复，在Git中，由于分支是如此的强大，所以，每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

当你接到一个修复一个代号101的bug的任务时，很自然地，你想创建一个分支issue-101来修复它，但是，等等，当前正在dev上进行的工作还没有提交：

并不是你不想提交，而是工作只进行到一半，还没法提交，预计完成还需1天时间。但是，必须在两个小时内修复该bug，怎么办？

git stash：可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作

git stash pop：恢复藏匿的内容到工作区，同时把stash内容也删了：

git stash apply：恢复藏匿的内容到工作区，（不删除stash中的内容）

git stash apply stash@{0}：恢复藏匿区指定的内容。

git stash drop stash@{0}：删除藏匿区指定的内容。

在master分支上修复了bug后，我们要想一想，dev分支是早期从master分支分出来的，所以，这个bug其实在当前dev分支上也存在。

那怎么在dev分支上修复同样的bug？重复操作一次，提交不就行了？

有木有更简单的方法？

有！

同样的bug，要在dev上修复，我们只需要把4c805e2 fix bug 101这个提交所做的修改“复制”到dev分支。注意：我们只想复制4c805e2 fix bug 101这个提交所做的修改，并不是把整个master分支merge过来。

为了方便操作，Git专门提供了一个cherry-pick命令，让我们能复制一个特定的提交到当前分支：

git cherry-pick 3044f93：复制bug分支的提交到当前分支，（3044f93 修复bug的提交id）

3.4 强行删除分支

git branch -D newbranch：强行删除分支

3.5 多人协作

推送分支

git push origin master：推送主分支

git push origin new：推送其他分支

并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

* master分支是主分支，因此要时刻与远程同步；
* dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；
* bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；
* feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

3.1合并方式：

Fast-forward：

这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。