0.初始化 到指定目录 git init

1.提交更新

1.1添加git add filename.txt filename2.txt

1.1.1 git add -u 只添加修改的文件

git add . ：他会监控工作区的状态树，使用它会把工作时的**所有变化提交**到暂存区，包括文件内容修改(modified)以及新文件(new)，但不包括被删除的文件。

git add -u ：他仅监控**已经被add的文件**（即tracked file），他会将被修改的文件提交到暂存区。add -u 不会提交新文件（untracked file）。（git add --update的缩写）

git add -A ：是上面两个功能的合集（git add --all的缩写）

1.2提交git commit -m “XXXXX”

2.查看版本信息

2.1查看 git log

2.1.1 一行显示结果 git log --pretty=oneline

2.1.2 行数过多可通过箭头、PageUp、PageDown等键滚动，q退出

2.1.3 显示确定条目 git log -n

3.版本回退

3.1回退到上一版本 git reset--hard HEAD^

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交3628164...882e1e0（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100

3.2 回退到任一版本并更新到工作区 git reset --hard <commit id>

3.3 查看历史操作，获取<commit id> git reflog

4.工作区、暂存区和版本库

4.1 工作目录为工作区，add操作后文件在暂存区，commit后提交到版本库，暂存区清空

4.2 查看暂存区当前状态git status

状态-working tree clean 干净

Untracked files 存在未添加（add）到暂存区的文件

Changes not staged for commit 文件修改后未添加（add）到暂存区

Changes to be committed 文件添加（add）到暂存区后未提交（commit）

4.3 查看具体区别git diff

4.3.1 git diff 查看工作区和暂存区的区别

4.3.2 git diff --cached 查看暂存区和版本库之间的区别

4.3.3 git diff HEAD 查看工作区和版本库之间的区别

4.4 查看提交后的变化git show <id>

5.撤销修改

5.1撤销工作区修改（用暂存区或版本库的最新版本覆盖工作区文件）

git checkout -- <filename>

暂存区的内容会被优先使用，如果需要清除暂存区内容，用5.2操作后，再用5.1

5.2 清除暂存区修改

git reset HEAD <filename>

6.删除

6.1 工作区删除 rm <filename>

6.2 暂存区删除 git rm <filename> 只有当工作区文件已删除后才能用（执行6.1后）

6.2.1 暂存区删除，工作区保留 git rm –cached <filename>

6.2.2 暂存区工作区同时删除 git rm –f <filename>

7.远程仓库

7.1 设置地址 git remote add origin ssh://git@g.hz.netease.com:22222/wangtao3/first-project.git

7.2 从远程仓库克隆 在上级目录 执行

git clone ssh://git@g.hz.netease.com:22222/wangtao3/first-project.git

7.2.1 从指定分支克隆

git clone -b <branch> ssh://git@g.hz.netease.com:22222/wangtao3/first-project.git

7.3 将远程存储库中的更改合并到当前分支中

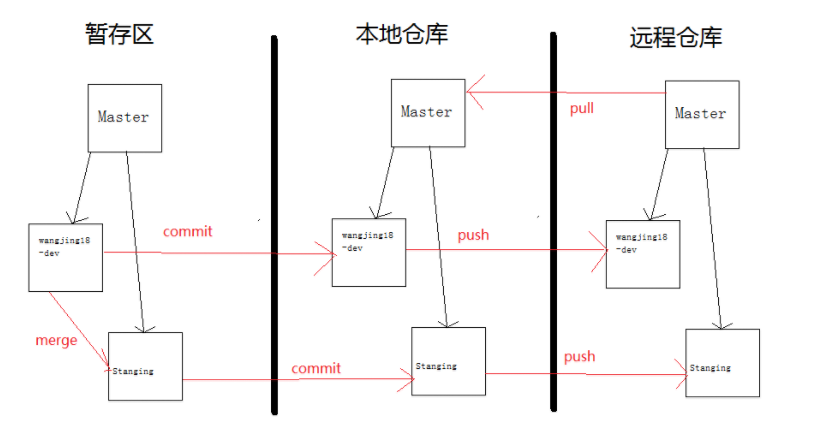
git pull <远程主机名> <远程分支名>:<本地分支名>

7.4 推送到远程

git push <远程主机名> <本地分支名>:<远程分支名>

7.5 删除远程分支

git push <远程主机名> :<远程分支名>



8.分支操作

8.1 查看分支 git branch

8.2 新建分支 git branch <branchname>

8.3 切换到分支 git checkout <branchname>

8.4 删除本地分支 git branch -d <branchname>

9.merge 和 rebase

例如现在有两个分支 master 和 feature, 你在 feature 分支上进行了实验，这时候有个另外的人在 master 分支上进行了新的提交。那么你需要将 master上别人的修改应用到 feature 分支上。

方法1： merge

git checkout feature

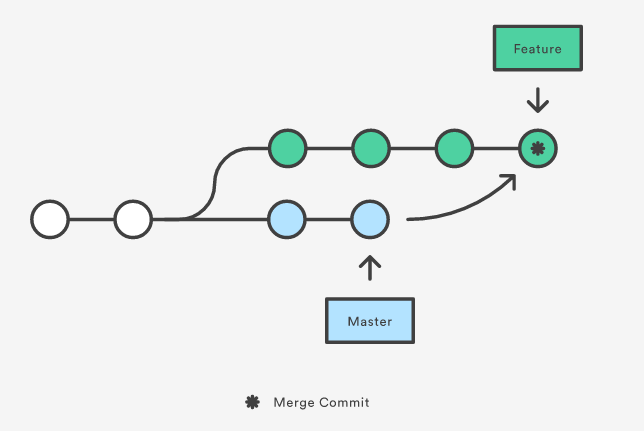
git merge master

# 或者直接指定两个分支：

git merge master feature

* 1
* 2
* 3
* 4
* 5

这样操作后会在 feature 分支上产生一个新的 commit, 这个commit就是包含了 master 分支的修改。同时历史记录中也会包含这个 commit 的信息。这样会有个好处，也会带来问题。   
**好处就是：merge 是 non-destructive 的操作，比较安全。（相对于rebase操作）**   
**问题就是：如果merge频繁，那么 feature 分支的历史记录中会包含很多个由于 merge 产生的新 commit 信息。这可能不是你希望看到了…**）。

如图：   


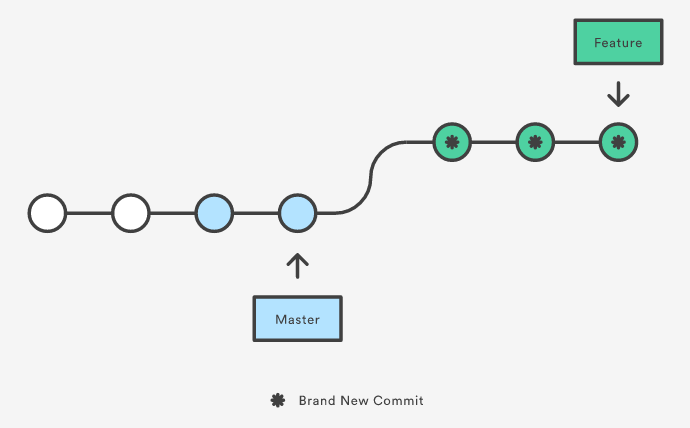
方法2： rebase

git checkout feature

git rebase master // 将 master 上的修改合并到 feature 分支（当前分支）。

* 1
* 2

rebase 会将 feature 上的历史 commit 全部修改，并且用新的提交覆盖之（即下图中的 **Brand New Commit**）。   
看起来就是你的 feature 分支从一个开始就是在最新的 master 上开发的( 新的master跑到了你分支的最开始处 )。

如图（注意比较与上图 merge 的差异）：   


一个注意点就是不要将 master 分支 rebase 到其他分支上面。这样会导致该 master 分支和其他人的 master 分支的历史记录不一样。然后你还得将你的 master 分支与别人的master分支merge。

所以，在 rebase 一个分支前想一下别人有没有 watch 这个分支， 因为rebase 会将该分支的历史提交修改。