切换前，Git 将会强制要求我处理 未提交的更改，要么提交，要么舍弃。

最擅长描述一个问题的人最有可能能够解决它。

创建项目克隆将会下载仓库中所有文件的副本及其提交历史，还会记住你下载代码的目录，将远程代码托管服务器设置为跟踪仓库。

可以在这个目录中随意折腾。如果你 把这个目录弄得面目全非，删除文件夹并重新运行 clone 命令即可。

你应该养成经常使用 status 命令的习惯，就像在文 字处理程序中使用保存那么频繁。

要查看仓库中产生的更改，请使用 log 命令。默认情况下，这个命令允许你查看 本地仓库当前签出分支中的每个提交的提交消息和作者信息。

git log



可以通过添加 --oneline 参数来缩短消息，只显示消息的第一行

git log --oneline



列出所有分支，可以使用不加参数的branch命令，或者添加—list参数。--all或者-a可以列出所有分支，包括远程分支。--remotes仅列出远程分支。

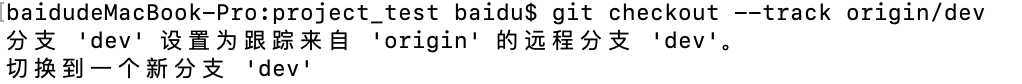
默认情况下，克隆会将远程的master分支复制到本地仓库，除了这个分支之外，还下载了远程仓库中所有其他已有的分支，在引用这些分支之前需要设置远程分支的工作副本。

远程分支列表不会自动更新，因此这个列表将会随着时间而落后。使用 fetch 命令更新这个列表

git fetch

使用 checkout 命令切换远程分支并在本地创建同名分支，并切换到新建的分支

git checkout --track origin/dev



或者使用git checkout --track -b video-lessons origin/video-lessons

创建一个新的开发分支：

首先切换到起点分支。git checkout master

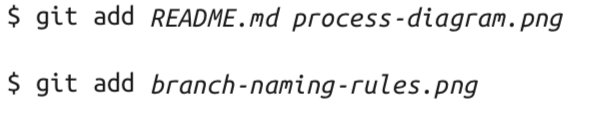
创建一个新分支 git branch 1-process\_notes

签出/切换到新分支 git checkout 1-process\_notes

或者使用下面的命令，可以直接从正确的基线分支上创建一个新的分支。

git checkout -b 1-process\_notes master

add提交多个修改文件时，(同目录)文件间空格隔开，也可以一个一个的add



通过使用 --update 参数，你可以暂存 Git 中所有已知的且在上次提交之后编辑过(或修改过)的文件

git add --update

在svn 中，一把有一个仓库的副本，所有工作都写入副本，在git中没有每个人共同使用单一仓库，当你一个人工作时，远程仓库更多的是作为你本地仓库的备份。

一旦代码经过完整的测试，就需要将这个工单分支并入master分支，并删除本地分支和工单分支的远程分支。

将本地工单分支并入本地主分支：

$ git checkout master

$ git merge 1-process\_notes

将合并后的master分支推送到远程仓库

$ git push

为了保持仓库整洁，可以删除本地的工单分支，删除本地分支：

git branch --delete 1-process\_notes

或者git branch -D 1-process\_notes

如果该分支没有并入其他分支，git会提醒

删除不再需要的远程分支：

git push --delete my\_gitlab 1-process\_notes

基本GIT命令：

dir 在win下列出文件

git clone URL 克隆一份远程仓库的副本

git init 把当前目录转换为成一个新的git仓库

git checkout --track remote\_name/branch 在本地创建远程分支的副本，本地分支同名

git checkout branch 切换到另一个本地分支

git checkout -b branch branch\_parent 从指定本地分支创建一个新分支，不指定从当前分支

git tag 列出所有标签

git show tag 输出所有带标签的详细信息

git remote add remote\_name URL 创建一个指向远程仓库的引用

git remote –verbose 列出所有可用远程连接中fetch 和push命令使用的URL

git push –set-upstream remote\_name branch\_local branch\_remote 将本地分支的副本推送远程服务器

git merge branch 将当前存储在另一个分支branch的提交并入当前分支。

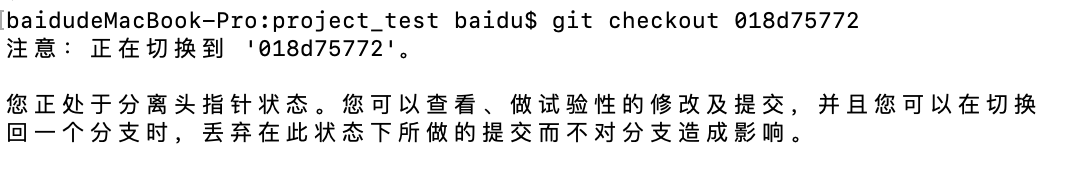
Git可以让你穿越回去，撤销已经犯下的错误，根据错误的严重程度，git提供不同的命令

分支变基，指使用父分支上提交来更新一个分支的过程，就是。。。。

Log命令显示了某个分支上的提交，reflog命令列出了仓库的本地副本完整的历史记录，从远程服务器克隆时就开始记录。

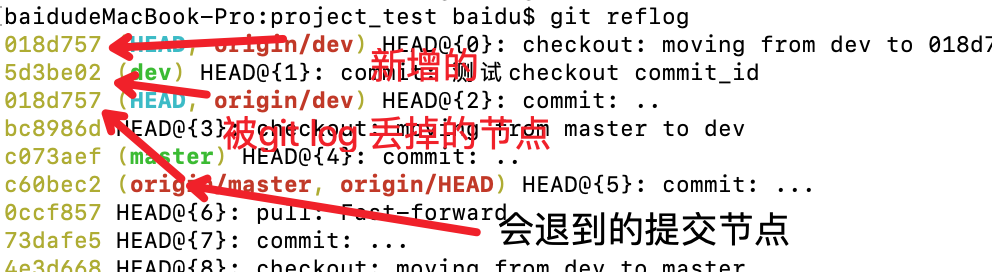
log 和 reflog 这两个命令都会显示仓库中某个状态的提交 ID。只要你找到了提交 ID，就可以签出这个提交，及时将代码库的版本临时恢复到那个节点。

$ git checkout commit\_id



注意：切换状态节点后，git log 显示的是会退到的节点和之前的提交，而git reflog 会新增一条记录

$ git reflog



$ git log



6.4还原文件

* 舍弃工作目录中对一个文件的修改（修改的文件未被暂存和提交）：git checkout -- <file>

例如：在工作区删除一个文件

$ rm README.md

此时未使用git add把删除动作（改动）提交到暂存区，可以使用下面命令舍弃工作区修改（同步工作区和本地库）

$ git checkout -- README.md

* 舍弃工作目录中所有未保存的变更（文件已暂存，未提交到本地库）：git rest hard <file>+ git checkout -- <file>

需要使用 reset 命令在恢复文件之前将它取消暂存，再git checkout；

例如：

$ rm README.md

$ git add README.md

使用 Git 的 HEAD(头指针)快捷方式，也就是指向当前分支最近一次 提交的指针，如下所示。

$ git reset HEAD README.md

文件被取消暂存后，可以像之前使用 checkout 命令那样恢复被删除的文件

$ git checkout -- README.md

注意：可以把上面两个命令合并成一个，如下所示。

$ git reset --hard HEAD -- README.md

总结：git 回滚

1．修改完，还未执行git add

git checkout .

或者

git checkout -- <file>

注意：使用暂存区的文件覆盖工作区，所以执行完git add .之后，再执行该命令是无效的

2.使用git add 提交到暂存区，还未commit之前

git reset HEAD <file> 先用Head指针覆盖当前的暂存区内容

git checkout . 或者 git checkout -- <file> 再用暂存区内容覆盖工作区内容

办法二：可以使用git reset --hard *origin/master*直接使用head覆盖当前暂存区和工作区

3．已经git commit，还未git push

办法一：从远程仓库把代码取回来，然后覆盖本地仓库、本地暂存区和工作区

git reset *--hard origin/master*

办法二：获取提交的commit\_id ,回退到上次提交，覆盖暂存区，再覆盖工作区

先查看提交的commit\_id，命令为

git log或者git log --oneline

覆盖本地的暂存区，

git reset --hard cf79123

覆盖本地工作区

git checkout .

4.已经git push

那就没办法了