# 1、时光机穿梭

## 1、版本回退

1、git log

显示从最近到最远的提交日志，如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数

2、回退到上一个历史版本（以下大写Git应为git）

Git reset --hard HEAD^

回退到上上个版本

Git reset --hard HEAD^^

回退到上上上个版本

Git reset --hard HEAD^^

回退到往上 100个版本

Git reset --hard HEAD~100

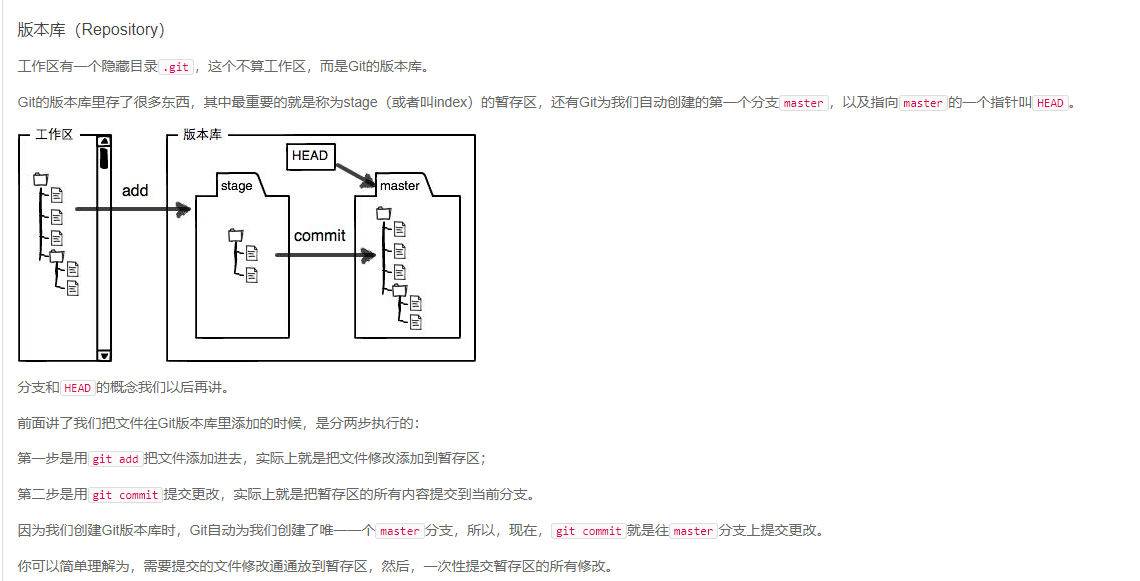
3、回退到未来某一个版本

Git reset --hard （加上commit id）

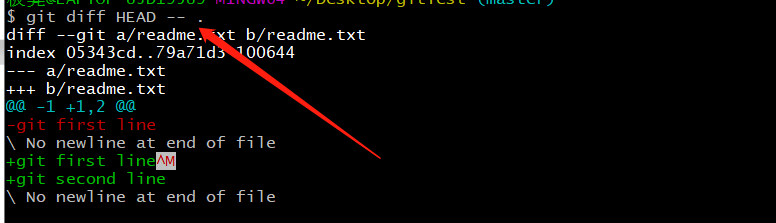
4、git reflog 命令

该命令用来记录你的每一次命令

## 2、工作区和暂存区



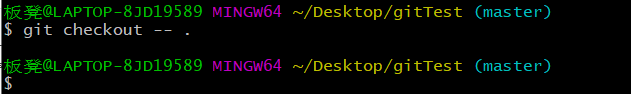
1、用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别



## 3、撤销修改

1、工作区修改完毕，但是还未提交

git checkout -- file可以丢弃工作区的修改



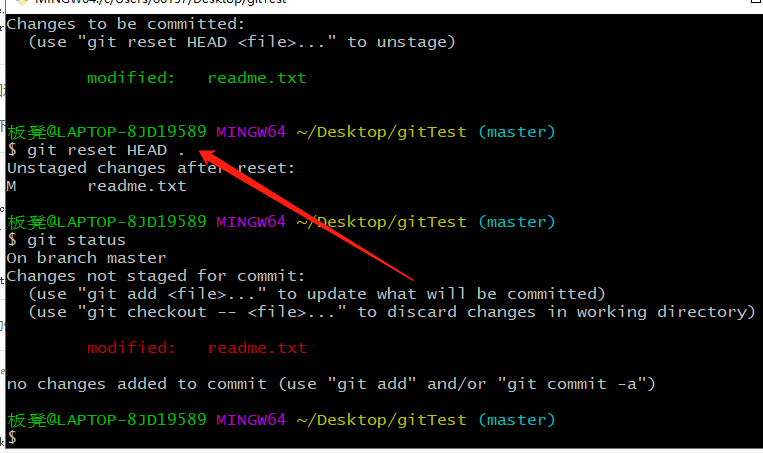
命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一、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；

二、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；

2、工作区修改完毕后提交到了暂存区，撤销暂存区的修改

用命令git reset HEAD <file>可以把暂存区的修改撤销掉（unstage），重新放回工作区



## 4、删除文件

1、工作区删除了文件，导致工作区和版本库中不一致

一、从版本库中也删除这个文件

确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit

git rm readme.txt

不要使用 git rm -r . （这样会递归删除所有文件）

二、删错了，恢复这个文件

使用命令从版本库恢复文件

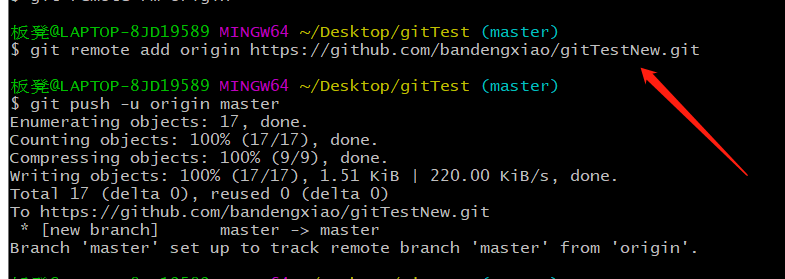
git checkout -- .

# 2、远程仓库

## 1、添加远程仓库

本地项目关联远程仓库

git remote add origin git@github.com:michaelliao/learngit.git



本地项目与远程仓库解除关联

git remote rm origin(删除关联的origin的远程库)

相关帖子：<https://blog.csdn.net/qq_40428678/article/details/84074207>

## 2、git首次推送本地分支到远程仓库

git push -u origin master

把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令。

# 3、分支管理

## 1、切换分支

创建dev分支，然后切换到dev分支：git checkout -b dev

$ git checkout -b dev

Switched to a new branch 'dev'

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

## 2、合并分支

把dev分支的工作成果合并到master分支上：

$ git merge dev

Updating d46f35e..b17d20e

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

## 3、删除分支

删除Dev分支

$ git branch -d dev

Deleted branch dev (was b17d20e).

## 4、使用switch切换分支

创建并切换dev分支

$ git switch -c dev

直接切换已有的dev分支

$ git switch dev

使用新的git switch命令，比git checkout要更容易理解。

## 5、解决冲突

git merge时报错有冲突，手动将冲突文件修改，可以使用cat命令查看冲突文件，修改完毕后git commit 解决完冲突

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。

用git log --graph命令可以看到分支合并图。

## 6、bug分支

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？

幸好，Git还提供了一个stash功能，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 stashSaved working directory **and** index state WIP on dev: f52c633 add merge

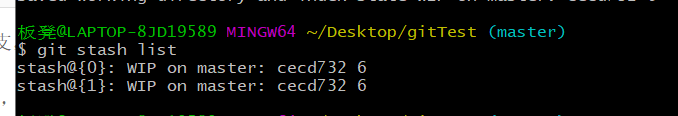
在别的分支修复完毕后回到原来的开发分支，用git stash list命令查看保存列表

一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；

另一种方式是用git stash pop，恢复的同时把stash内容也删了：

你可以**多次stash**，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：

$ git stash apply stash@{0}

### 复制特定提交到当前分支

适用情况：（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。

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 branch

\* dev

master$ git cherry-pick 4c805e2

[master 1d4b803] fix bug 101

1 file changed, 1 insertion(+), 1 deletion(-)

Git自动给dev分支做了一次提交，注意这次提交的commit是1d4b803，它并不同于master的4c805e2，因为这两个commit只是改动相同，但确实是两个不同的commit。用git cherry-pick，我们就不需要在dev分支上手动再把修bug的过程重复一遍。

## 7、强行删除分支

如果要丢弃一个没有被合并过的分支，可以通过git branch -D <name>强行删除。

## 8、多人协作

1、远程库信息，使用git remote -v；

2、本地新建的分支如果不推送到远程，对其他人就是不可见的；

3、从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；

4、在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

5、建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；

6、从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

## 9、操作远程分支

帖子：https://blog.csdn.net/cbmljs/article/details/88192868