Git版本管理

<https://github.com/doggy8088/Learn-Git-in-30-days/blob/master/docs/01%20%E8%AA%8D%E8%AD%98%20Git%20%E7%89%88%E6%9C%AC%E6%8E%A7%E7%AE%A1.markdown>

Git: distributed version control system (DVCS)

# 1. 建立储存库

在本机，建立本地储存库local repository

在本机，建立公用储存库shared repository

在GitHub或其他Git平台建立远端储存库remote repository

## 1.1 Local repository

Git Shell > mkdir git-demo > cd git-demo

这是空目录，不包含任何Git repository，用git init建立储存库，预设储存库会放在工作目录下的.git下

## 1.2 Shared repository

Shared repository指建立一个Git储存库，但不包含工作目录，常发生在Linux下，因为Linux通常多人使用同一台主机。用git init –bare建立，会在当前目录建立所有Git储存库的相关档案与资料夹。这个资料夹不能直接编辑，透过git指令操作。因为“没有工作目录的纯储存库”，也叫bare repository

Git是“分布式版本管控”，每人都有一份完整的repository。当你想建立一个工作目录working directory时，必须先取得这个bare repository内容回来：git clone [REPO\_URL]

建立工作目录，而且把这个shared repository完整地复制回来。

**注意**：工作目录下，.git也是一个储存库，但是工作目录下.git还包含工作目录下的索引资讯，记录工作目录下的状态、信息，这些不会出现在shared repository中。Shared repository里只有版本资讯。

## 1.3 Remote repository

可以在网站上完成相关操作，也可以使用command line

touch README.md

git init

git add README.md

git commit –m “first commit”

git remote add origin <https://github.com/cqbaizuo/LearnGit.git>

git push –u origin master

# 2. 常用的Git版本管控指令

[master +10 ~1 -2!]

master表示目录工作目录是master 分支，也是Git默认分支

“**红色**”都代表Untracked的档案，即这些变更都不会进入版本管控

+10 新增10个文件

~1 修改1个文件

-2 删除2个文件

## 2.1 新增档案

git add . 告知Git，这些变化“即将”加入Git版本库，但还没建立任何版本

git add app: 加入app资料夹与其下的所有档案

git add .\*: 加入所有”.”开头的档案

变成“**绿色**”，staged，变更会进入版本管控

## 2.2 提交变更、建立版本

git commit –m “words to explain this version”

git help commit git help add

## 2.3 查询历史记录

git log git log -10

## 2.4 档案更名

git mv test unit-test

## 2.5 显示工作目录的状态

git status git status –s: 精简版本

## 2.6 重置当前工作目录

git reset: 只会重置“索引”状态，用git rm删除的目录、文件，git mv更名的目录、文件，不能通过git reset把实体档案找回来

如果想把工作目录还原到目前的最新版，git reset --hard

## 2.7 还原其中一个档案

git checkout master Gruntfile.js 把master分支最新版的Gruntfile.js还原，可以避免使用git reset --hard 将所有档案还原

# 3. 了解储存库、工作目录、物件、索引之间的关系

## 3.1 储存库

### 3.1.1 了解储存库

使用Git进行版本，需要一个“版本库”来储存这些版本信息，这就是Repository

### 3.1.2了解工作目录

$ mkdir git-demo

$ cd git-demo

$ git init

git-demo自动地成为了“工作目录”working directory，我们正在开发的专门文件夹，未来的操作都在这个目录下进行。我们操作Git相关指令、参数时，通常也在working directory下执行。

使用Git版本管控时，会遇到很多分支。适时地保持工作目录干净，是版本管控的一个基本原则。

## 3.2 了解Git的资料结构

Git里两个重要的资料结构：物件、索引。物件：保存版本库中文件与版本记录；索引：保存当下要进版本库之前的目录状态

### 3.2.1 关于物件

物件：特别的档案，内容取出，产生一组SHA1杂凑值，命名这个档案。

物件分为：目录资讯和档案内容，即tree object & blob object，都存放于object storage中。

### 3.2.2 关于索引

“索引”是一个经常异动的暂存档，通常位于.git目录下，名为index的档案。“索引”，记录“哪些档案即将要被提交到下一个commit版本中”，想要提交一个版本到Git repository，那么一定要先更新索引状态。

通常不直接编辑.git\index这个二进位档，而是透过git指令：

* git add
* git mv, git rm
* git status
* git commit
* git ls-files

索引：位于working directory和object storage之间的媒介。任何working directory的变化，都需要先更新索引，才能加入object storage版本管控中。

git branch rename: git branch -m <oldname> <newname> git branch -m <newname>, for the current branch

# 4. 物件结构

物件是immutable，所有储存在object storage中的档案，通常只出不进。

## 4.1 物件资料库

储存在.git\objects中，每个物件根据“档案内容”，进行SHA1杂凑运算，得到hash值，作为文档名称。

## 4.2 物件名称

* blob物件：某个档案的“内容”，并且只是内容而已。执行git add指令时，这些新增档案，立刻被写入成为blob物件，名称是杂凑值。
* tree物件：某个资料夹的快照snapshot
* commit物件：特定版本有哪些tree物件、版本提交时间、记录信息，通常还有上一层commit物件名称
* tag物件

# 5. 索引结构

## 5.1 关于索引

索引：记录“哪些档案即将被提交到下一个commit版本中”。想要提交一个版本到Git repository，必须先更新索引状态git add。有很多别名：index, cache, directory cache, current directory cache, staging area, staged files

如：git diff --cached == git diff --staged



$ git status:

# changes to be committed

# new file: c.txt

# changes not staged for commit:

# modified: a.txt

# untracked files

# b.txt

* changes to be committed
  + 最新版没有c.txt
  + 索引档加入了c.txt
  + git commit后，存入下一个版本
* changes not staged for commit
  + 最新版有a.txt
  + 索引档没加入a.txt
  + git commit不会进入下一版本
* untracked files
  + 最新版没有b.txt
  + 索引档也没有b.txt
  + git commit不会进入下一版本

实际上，git commit是将索引档里的文件加入创建的版本中。因此，只有目前最新版和索引档之间有差异的变更，才会被提交到下一个commit物件中

## 5.2 常用命令

* git add
* git rm git rm --cached a.txt 删除索引档中该文档，但保留实体文件
* git mv
* git commit
* git ls-files

# 6. 分支的具体观念和使用方式

## 6.1 分支的基本观念

最主要的目的：开发过程中，版本冲突的问题，可以：从远端pull一个分支到本地分支；将本地分支push到远端；也可以将本地任何一个分支merge到本地另一个分支

## 6.2 常用操作

* git branch NewBranch 创建branch，并停留在当前branch
* git checkout -b NewBranch 创建branch，切换到新的branch
* git branch 查看现有branch
* git checkout BranchName 查看BranchName
* git branch -d BranchName 删除BranchName
* git checkout CommitID 查看某个commitID

**注意**：