1. Git的诞生。
2. 集中式（CVS及SVN）vs分布式（git）
3. 集中式（版本控制系统）：版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。

（不足：必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟）

1. 分布式（版本控制系统）：没有“中央服务器”、每个人的电脑上都是一个完整的版本库、工作时不需要联网，因为版本库就在你自己的电脑上。

（优点：安全性高，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。）

1. 创建版本库

1、创建版本库

$ mkdir learngit

$ cd learngit

$ pwd // pwd命令用于显示当前目录

$ git init // git init命令把这个目录变成Git可以管理的仓库

1. 把文件添加到版本库

$ git add file

$ git commit -m "xxx"

1. git基本操作

1、git status //可以让我们时刻掌握仓库当前的状态

2、git diff file //可以从命令输出看到作了什么修改

3、版本回退

git log //显示从最近到最远的提交日志

例如：得到1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master) append GPL

这里1094adb...的是commit id（版本号）

(HEAD -> master) append GPL最近的一次提交是append GPL

git reset （--hard HEAD^） //回退（上一个版本）

备注：

在Git中，用HEAD表示当前版本、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

git reset --hard （1094a） //通过commit id（版本号）可以再次返回到最新的版本，，也可以到达任一版本、同时版本号可以不用写全只用写最前几位数字就行

备注：

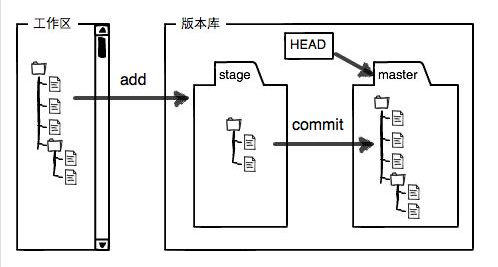
如果回退中电脑关闭了，可以通过$ git reflog命令查看你的每一次命令记录，可通过记录回退到最新版本

1. 工作区和暂存区

工作区（Working Directory）：你在电脑里能看到的目录

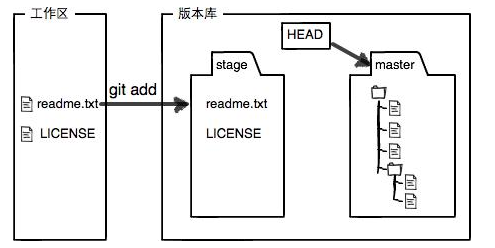
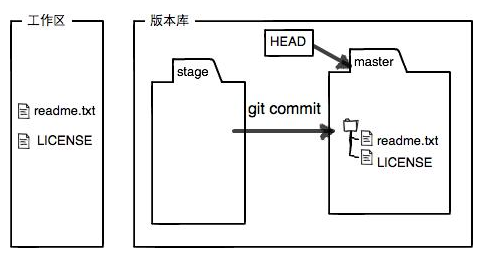
版本库（Repository）：工作区有（一个隐藏目录）.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。



（1）、git add把文件添加进去，实际上就是把文件修改添加到暂存区；

（2）、git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支

1. 撤销修改

（1）、git add 之前

git checkout -- file 可以丢弃工作区的修改：git checkout -- file命令中的“--”很重要，没有“--”，就变成了“切换到另一个分支”的命令

存在两种状态：

一种是file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；

一种是file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。

总之，就是让这个文件回到最近一次git commit或git add时的状态。

（2）git add提交到缓存区之后之后

git reset HEAD <file> //git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。

1. 删除文件;

$ git rm file

1. 远程仓库(本地Git仓库和GitHub仓库之间的传输是通过SSH加密的)

1、确定是否有.ssh（id\_rsa和id\_rsa.pub）

2、创建：$ ssh-keygen -t rsa -C ["youremail@example.com"](mailto:\"youremail@example.com\")

列如：在GitHub创建一个仓库learngit

然后关联远程仓库：$ git remote add origin [git@github.com:Time-Passing(GitHub账户名)/learngit.git](mailto:git@github.com:Time-Passing(GitHub账户名)/learngit.git)（初始origin）

1. 把本地库的所有内容推送到远程库上

$ git push -u origin master（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。）

1. 从远程库克隆

$ git clone [git@github.com:Time-Passing(GitHub账户名)/learngit.git](mailto:git@github.com:Time-Passing(GitHub账户名)/learngit.git)

$ git clone <https://github.com/Time-Passing(GitHub账户名)/learngit.git>

使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令

1. 分支管理

1、

创建并切换分支（创建指针）：$ git checkout -b dev == $ git branch dev $ git checkout dev === git switch -c dev

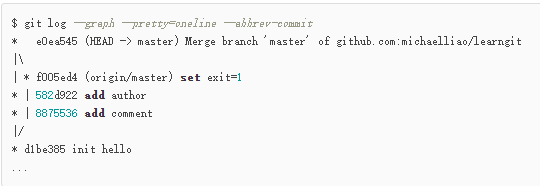
查看分支;git branch

查看远程库的信息，用git remote/git remote -v;(显示更详细的信息)

合并分支：$ git merge dev

删除分支：$ git branch -d dev

2、Rebase(变基)（可以把本地未push的分叉提交历史整理成直线）



$ git rebase之后



原理：Git把我们本地的提交“挪动”了位置，放到了f005ed4 (origin/master) set exit=1之后，这样，整个提交历史就成了一条直线。rebase操作前后，最终的提交内容是一致的，但是，我们本地的commit修改内容已经变化了，它们的修改不再基于d1be385 init hello，而是基于f005ed4 (origin/master) set exit=1，但最后的提交7e61ed4内容是一致的。

这就是rebase操作的特点：把分叉的提交历史“整理”成一条直线，看上去更直观。缺点是本地的分叉提交已经被修改过了。

#### 标签管理（通过tag确定了打标签时刻的版本。将来无论什么时候，都能取某个标签的版本）

1、切换到需要打标签的分支上；

2、git tag <name>打一个新标签：（$ git tag v1.0）；

3、git tag查看所有标签

4、对历史版本打标签：$ git tag v0.9 f52c633 //（commit id：f52c633）

5、git show <tagname>查看标签信息标签不是按时间顺序列出，而是按字母排序的

6、创建带有说明的标签，用-a指定标签名，-m指定说明文字：

$ git tag -a v0.1 -m "version 0.1 released" 1094adb

7、删除标签：$ git tag -d v0.1

8、推送某个标签到远程，使用命令git push origin <tagname>

一次性推送全部尚未推送到远程的本地标签：git push origin --tags

1. 删除远程标签：

先删除本地标签：$ git tag -d v0.9

然后，从远程删除：$ git push origin :refs/tags/v0.9

登陆GitHub查看是否真的从远程库删除了标签