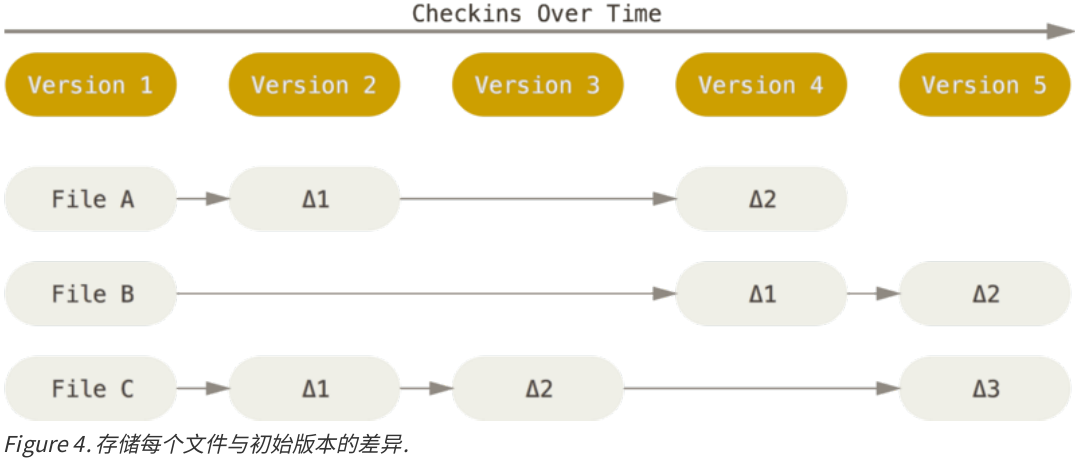
# 第1章 起步

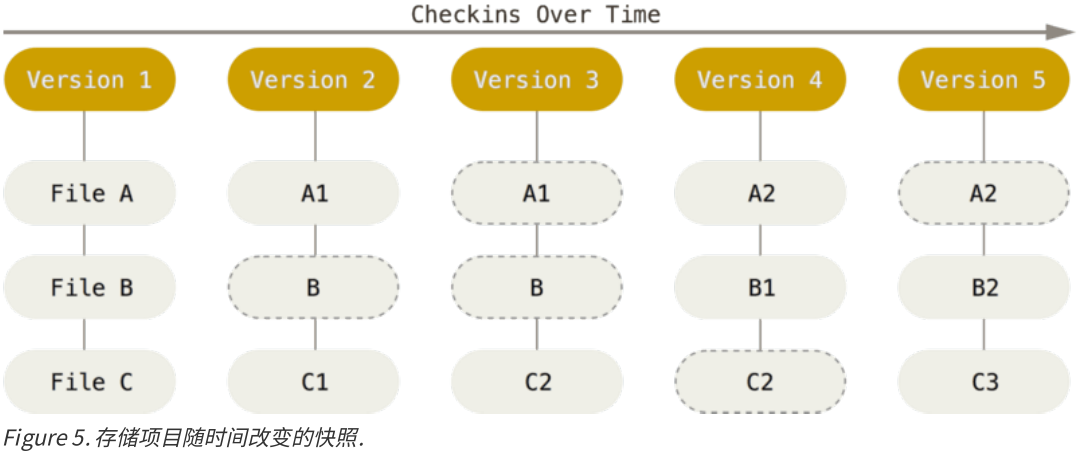
## 1.1 Git基础

### 1.1.1 直接记录快照，而非差异比较

Git和其它版本控制系统的主要差别在于Git对待数据的方法。概念上来区分，其它发部分系统以文件变更列表的方式存储信息。这类系统将它们保存的信息看作是一组基本文件和每个文件随时间逐步累积的差异。



Git不按照以上的方式对待或保存数据。反之，Git更像是把数据看做对小型文件系统的一组快照。每次提交更新，或在Git中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。为了更高效，如果文件没有修改，Git不再重新存储该文件，而是只保留一个链接指向之前存储的文件。Git对待数据更像是一个快照流。



### 1.1.2 近乎所有操作都是本地执行

在Git中的绝大多数操作都只需要访问本地文件和资源，一般不需要来自网络上其它计算机的信息。因为在本地磁盘上喊就有项目的完整历史，所以大部分操作看起来瞬间完成。如：要浏览项目的历史，Git不需要外连服务器去获取历史，然后再显示出来—它只需要直接从本地数据库中读取。

### 1.1.3 Git保证完整性

Git中所有数据在存储前都计算校验和，然后以校验和来引用。这意味着不可能在Git不知情时更改任何文件内容或目录内容。

Git用以计算校验和的机制叫做SHA-1散列。这是一个由40个十六进制字符组成字符串，基于Git中文件的内容或目录结构计算出来。

Git数据库中保存的信息都是以文件内容的哈希值来索引，而不是文件名。

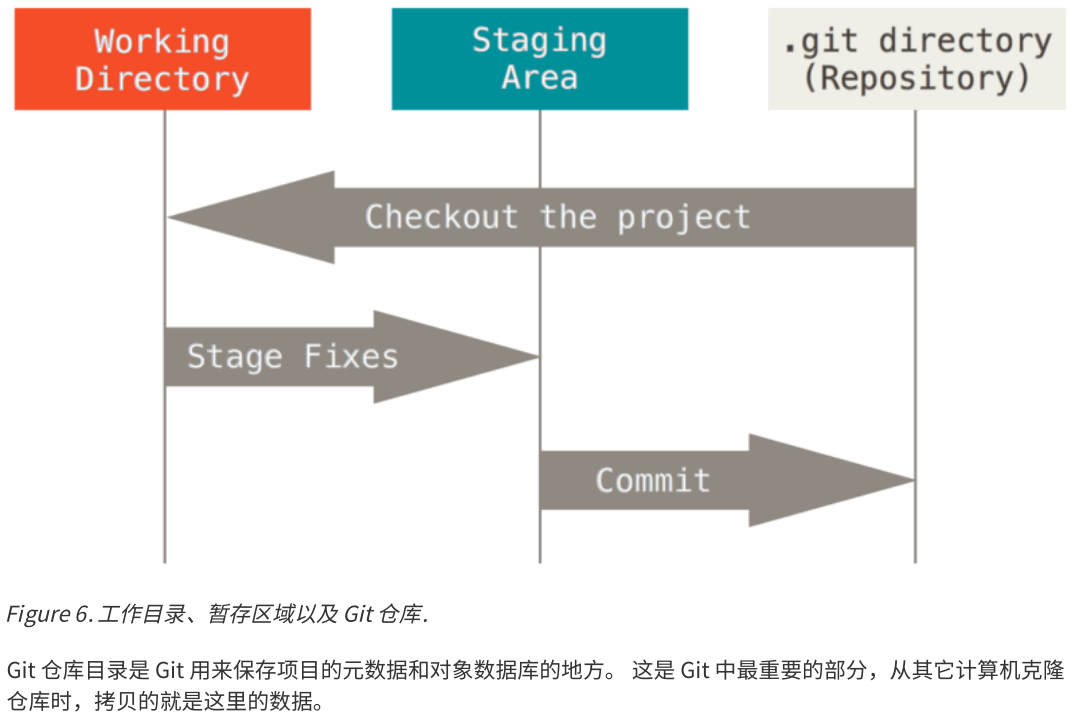
### 1.1.4 Git一般只添加数据

执行Git操作，几乎只往Git数据库中增加数据。很难让Git执行不可逆操作，或者让它以任何方式清除数据。

### 1.1.5 三种状态

Git有三种状态：已提交（committed）、已修改（modified）和已暂存（staged）。已提交表示数据已经安全的保存在本地数据库中；已修改表示修改了文件，但还没保存到数据中；已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

由此引入Git项目的三个工作区域的概念：Git仓库、工作目录以及暂存区域。



Git仓库目录是Git用来保存项目的元数据和对象数据库的地方。这是Git中最重要的地方，从其它计算机克隆仓库时，拷贝的就是这里的数据。

工作目录是对项目的某个版本独立提取出来的内容。这些从Git仓库的压缩数据库中提取出来的文件，放在磁盘上供使用或修改。

暂存区域是一个文件，保存了下次将提交的文件列表信息，一般在Git仓库目录中。

基本的Git工作流程如下：

（1）在工作目录中修改文件。

（2）暂存文件中，将文件的快照放入暂存区域。

（3）提交更新，找到暂存区域的文件，将快照永久性存储到Git仓库目录。

如果Git目录中保存着的特定版本文件，就属于已提交状态；如果作了修改并放入暂存区域，就属于已暂存状态。如果自上次取出后，作了修改但没有放到暂存区域，就是已修改状态。

## 1.2 初次运行Git前的配置

Git自带一个git config的工具来帮助设置控制Git外观和行为的配置变量。这些变量存储在三个不同的位置：

（1）/etc/gitconfig文件：包含系统上每一个用户及他们仓库的通用配置。如果使用带有--system选项的git config时，它会从此文件读写配置变量。

（2）~/.gitconfig或~/.config/git/config文件：只针对当前用户。可以传递--global选项让Git读写此文件。

（3）当前使用仓库的Git目录中的config文件（就是.git/config）：针对该仓库。

每一个级别覆盖上一级别的配置，所以.git/config的配置变量会覆盖/etc/gitconfig中的配置变量。

### 1.2.1 用户信息

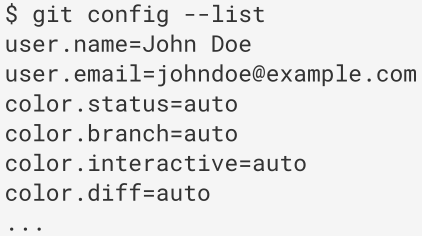
当安装Git应该做的第一件事就是设置用户名称和邮件地址。这样做很重要，因为每一个Git的提交都会使用这些信息，并且它会写入到每一次提交中，不可更改：



如果使用了--global选项，那么该命令只需要运行一次，因为之后无论在该系统上做任何事情，Git都会使用那些信息。当想针对特定项目使用不同的用户名称与邮件地址时，可以在那个项目目录下运行没有--global选项的命令来配置。

### 1.2.2 检查配置信息

如果想要检查配置，可以使用git config --list命令来列出所有Git当时能找到的配置。



可能会看到重复的变量名，因为Git会从不同的文件中读取同一个配置（如：/etc/gitconfig与~/.gitconfig）。这种情况下，Git会使用它找到的每一个变量的最后一个配置。

可以通过输入git config <key>：来检查Git的某一项配置：



# 第2章 Git基础

## 2.1 获取Git仓库

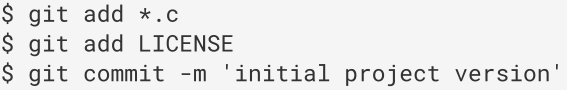
### 2.1.1 在现有目录中初始化仓库

如果打算使用Git来对现有的项目进行管理，只需要进入该项目目录并输入：



该命令将创建一个名为.git的子目录，这个子目录含有初始化的Git仓库中所有的必须文件，这些文件是Git仓库的骨干。但是在这个时候，仅仅是做了一个初始化的操作，项目里的文件还没有被跟踪。

如果是一个已经存在文件的文件夹（而不是空文件夹）中初始化Git仓库进行版本控制，那么可通过git add命令来实现对指定文件的跟踪，然后执行git commit提交：



### 2.1.2 克隆现有的仓库

如果想要获得一份已经存在了的Git仓库的拷贝，这时就要用到git clone命令。Git克隆的是该Git仓库服务器上的几乎所有数据，而不是仅仅复制完成工作所需要的文件。当执行git clone命令的时候，默认配置下远程Git仓库中的每一个文件的每一个版本都将被拉去下来。如果服务器的磁盘坏了，通常可以使用任何一个克隆下来的用户端来重建服务器上的仓库（虽然可能会丢失某些服务器端的挂钩设置，但是所有版本的数据仍存在）。

克隆仓库的命令格式：git clone [url]。



这回在当前目录下创建一个名为“libgit2”的目录，并在这个目录下初始化一个.git文件夹，从远程仓库拉取下所有数据放入.git文件夹，然后从中读取最新版本的文件拷贝。

如果想在克隆远程仓库的时候，自定义本地仓库的名字，可以使用如下命令：

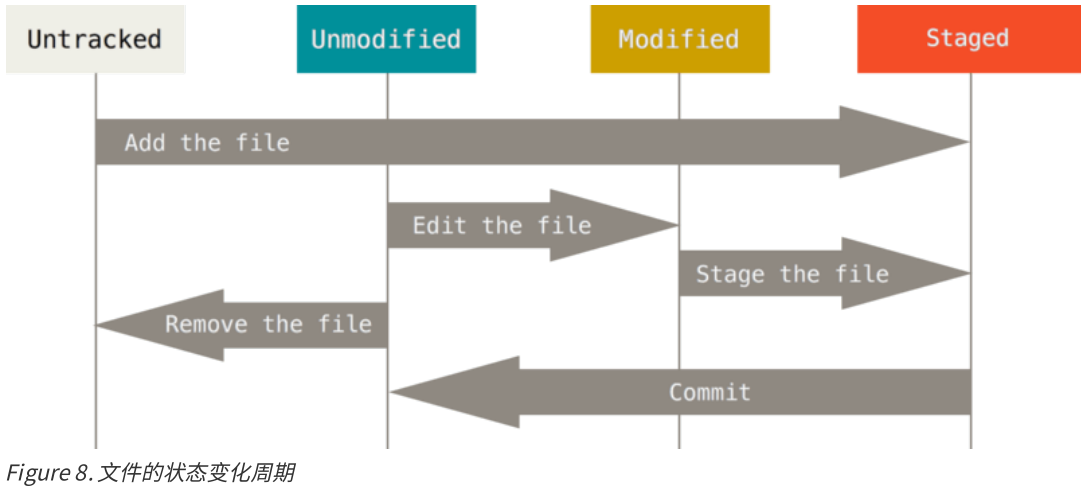


这将执行与上一个命令相同的操作，不过在本地创建的仓库名字变为mylibgit。

## 2.2 记录每次更新到仓库

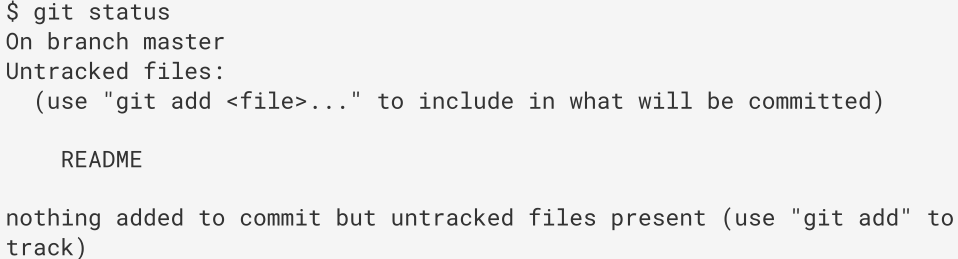
工作目录下的每一个文件都不外乎这两种状态：已跟踪和未跟踪。已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后，它们的状态可能处于未修改，已修改或已放入暂存区。初次克隆某个仓库的时候，工作目录中所有文件都属于已跟踪文件，并处于未修改状态。

编辑过某些文件之后，由于自上次提交后对它们做了修改，Git将它们标记为已修改文件，这些文件将被放入暂存区，然后提交所有暂存了的修改，如此反复。所有使用Git时文件的生命周期如下：

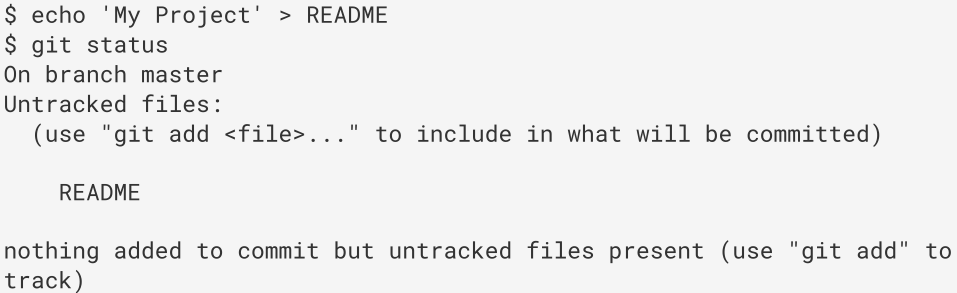


### 2.2.1 检查当前文件状态

要检查哪些文件处于什么状态，可以用git status命令。



在项目下创建一个新的README文件。如果之前不存在这个文件，使用git status命令将看到一个新的未跟踪文件：



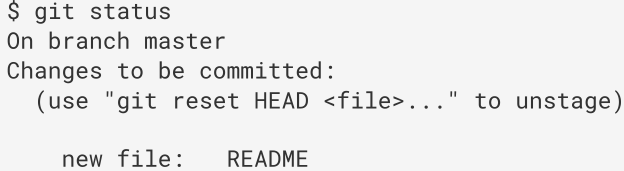
在状态报告中可以看到新建的README文件出现在Untracked files下面。未跟踪的文件意味着Git在之前的快照（提交）中没有这些文件；Git不会自动将之纳入跟踪范围，除非告诉Git“我需要跟踪这个文件”，这样的处理可以避免将生成的二进制文件或其它不想被跟踪的文件包含进来。

### 2.2.2 跟踪新文件

使用命令git add开始跟踪一个文件：



此时再运行git status命令，会看到README文件已被跟踪，并处于暂存状态：



只要在Change to be commited这行下面的，就说明是已暂存状态。如果此时提交，那么该文件此时此刻的版本将被留存在历史记录中。

Git add命令使用文件或目录的路径作为参数；如果参数是目录的路径，该命令将递归地跟踪该目录下的所有文件。

### 2.2.3 暂存已修改文件

修改一个已被跟踪的文件CONTRIBUTEING.md，然后运行git status命令：



文件CONTRIBUTEING.md出现在Changes not staged for commit这行下面，说明已跟踪文件的内容发生了变化，但还没有放到暂存区。要暂存这次更新，需要运行git add命令。这是个多功能命令：可以用它开始跟踪新文件，或者把它已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等。可以将这个命令理解为“添加内容到下一次提交中”而不是“将一个文件添加到项目中”要更加合适。

运行git add将“CONTRUBUING.md”放到暂存区，然后再看看git status的输出：



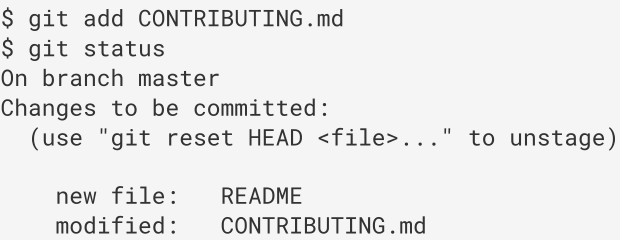
现在两个文件都已暂存，下次提交时就会一并记录到仓库。

但假设此时修改“CONTRUBUING.md”文件，添加一条注释后，再运行git status时：



出现“CONTRUBUING.md”文件同时出现在暂存区和非暂存区。实际上Git只是暂存了运行git add命令时的版本，途观现在提交，“CONTRUBUING.md”的版本是最后一次运行git add命令时的那个版本，而不是运行git commit时，在工作目录中的当前版本。所以，运行了git

add之后又作了修订的文件，需要重新运行git add把最新版本重新暂存起来：



注：运行了git add之后又修订了文件，需要重新运行git add把最新版本重新暂存起来。

### 2.2.4 状态简览

Git status命令的输出十分详细，但其用语有些繁琐。如果使用git status -s命令或gitstatus –short命令，将得到一种更为紧凑的格式输出。运行git status -s，状态报告输入如下：

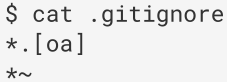


新添加的未跟踪文件前面有??标记；新添加到暂存区中的文件前面有A标记，修改过的文件前面有M标记。

M有两个可以出现的位置，出现在右边的M表示该文件被修改了但是还没放入暂存区，出现在靠左边的M表示该文件被修改了并放入了暂存区。

### 2.2.5 忽略文件

一般总会有些文件无需纳入Git的管理，也不希望它们总出现在未跟踪文件的列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。在这种情况下，可以创建一个名为. gitignore的文件，列出要忽略的文件模式。



第一行告诉Git忽略所有以.o或.a结尾的文件。一般这类对象文件和存档文件都是编译过程中出现的；第二行告诉Git忽略所有以波浪线（~）结尾的文件。

文件.gitignore的格式规范如下：

（1）所有空行或者以 ＃ 开头的行都会被 Git 忽略。

（2）可以使用标准的 glob 模式匹配。

（3）匹配模式可以以（/）开头防止递归。

（4）匹配模式可以以（/）结尾指定目录。

（5）要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

注：GitHub有一个十分详细的针对数十种项目及语言的.gitignore文件列表，你可以在https://github.com/github/gitignore找到它。

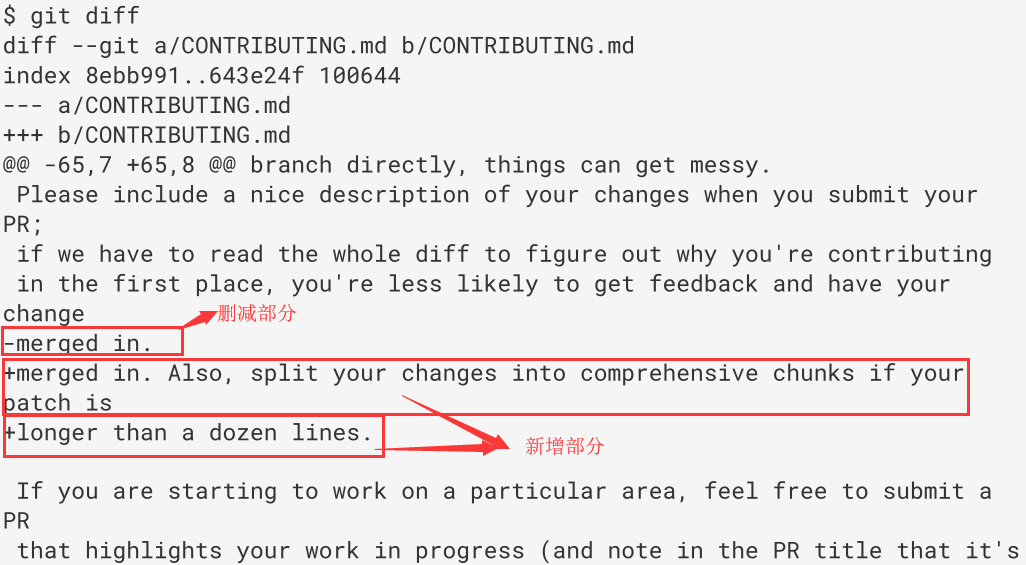
### 2.2.6 查看已暂存和未暂存的修改

如果git status命令的输出觉得过于模糊，想知道具体修改了什么地方，可以用git diff命令。

假如再次修改README文件后暂存，然后编辑CONTRIBUTING.md文件后先不暂存，运行 git status命令：



要查看尚未暂存的文件更新了哪些部分，不加参数直接输入git diff：



此命令比较的是工作目录中当前文件和暂存区域快照之间的差异，也就是修改之后还没有暂存起来的变化内容。

若要查看已暂存的将要添加到下次提交里的内容，可以用git diff –cached命令。

注：Git 1.6.1及更高版本还允许使用git diff --staged，效果是相同的。



注：git diff本身只显示尚未暂存的改动，而不是自上次提交以来所做的所有改动。所以有时候一下子暂存了所有更新过的文件后，运行git diff后却什么也没有，就是这个原因。

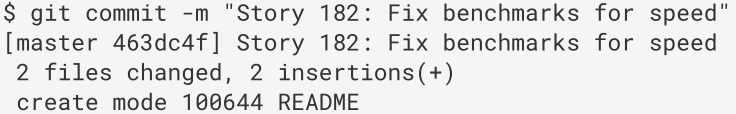
### 2.2.6 提交更新



提交更新的只会是那些已经暂存起来的文件，没有git add暂存的文件，是只会保留在本地磁盘，所以每次准备提交前，先用git status查看，是不是都已暂存起来了，然后再运行提交命令git commit。

默认的提交消息包含最后一次运行git status的输出，放在注释行里，另外开头还有一空行，供输入提交说明。可以去掉这些注释行，不过留着也没关系，多少能帮回想起这次更新的内容有哪些。

也可以在 commit 命令后添加 -m 选项，将提交信息与命令放在同一行：



注：提交时记录的是放在暂存区域的快照。任何还未暂存的仍然保持已修改状态，可以在下次提交时纳入版本管理。每一次运行提交操作，都是对项目作一次快照，以后可以回到这个状态，或者进行比较。

### 2.2.7 跳过使用暂存区域

Git提供了一个跳过使用暂存区域的方式，只要在提交的时候，给git commit 加上-a选项，Git就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过git add步骤。

### 2.2.8 移除文件

要从Git中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交。可以用git rm命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

如果删除之前修改过并且已经放到暂存区域的话，则必须要用强制删除选项-f（译注：即 force 的首字母）。这是一种安全特性，用于防止误删还没有添加到快照的数据，这样的数据不能被Git恢复。

另外一种情况是，想把文件从Git仓库中删除（亦即从暂存区域移除），但仍然希望保留在当前工作目录中。即：想让文件保留在磁盘，但是并不想让Git继续跟踪。当忘记添加.gitignore文件，不小心把一个很大的日志文件或一堆 .a这样的编译生成文件添加到暂存区时，这一做法尤其有用。为达到这一目的，使用--cached 选项：

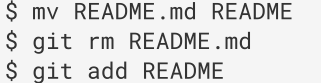


git rm命令后面可以列出文件或者目录的名字，也可以使用glob模式。

### 2.2.9 移动文件



其实，运行git mv就相当于运行了下面三条命令：



## 2.3 查看提交历史