**Git基本操作**

## Git工作流程

一般工作流程如下：

* 克隆 Git 资源作为工作目录。
* 在克隆的资源上添加或修改文件。
* 如果其他人修改了，你可以更新资源。
* 在提交前查看修改。
* 提交修改。
* 在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。

下图展示了 Git 的工作流程：

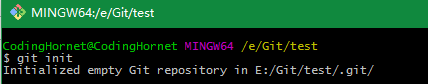


## 创建版本库

什么是版本库呢？版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

tips：如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

**git init**



瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

## 把文件添加到版本库

### 编写一个readme.txt文件，内容如下：

Git is a version control system.

Git is free software.

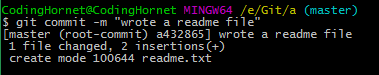
### 用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt



### 用命令git commit告诉Git，把文件提交到仓库：

$ git commit -m "wrote a readme file"



**-m**后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件，比如：

$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

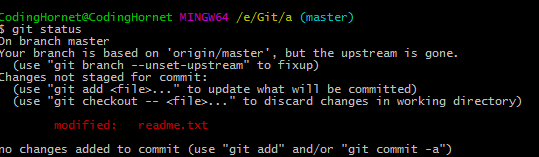
## 状态查看

### 修改readme.txt文件，改成如下内容：

Git is a distributed version control system.

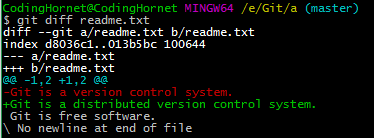
Git is free software.

### 运行git status命令看看结果：



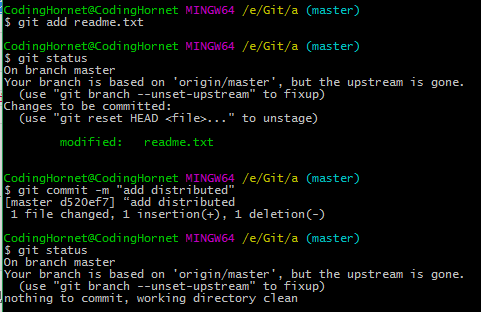
**git status**命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，readme.txt被修改过了，但还没有准备提交的修改。

### 虽然Git告诉我们readme.txt被修改了，但如果能看看具体修改了什么内容，自然是很好的。需要用git diff这个命令看看：



**git diff**顾名思义就是查看difference，显示的格式正是Unix通用的diff格式，可以从上面的命令输出看到，我们在第一行添加了一个“distributed”单词。

知道了对readme.txt作了什么修改后，再把它提交到仓库就放心多了。



**git status**告诉我们，将要被提交的修改包括readme.txt，下一步，就可以放心地提交了：

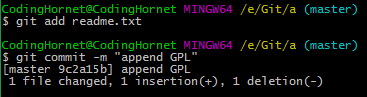
## 版本回退

### 修改readme.txt文件如下：

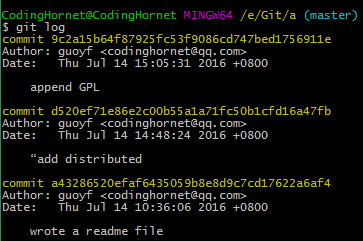
Git is a distributed version control system.

Git is free software distributed under the GPL.

然后提交：

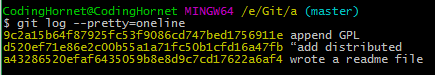


### 用git log命令查看日志：



**git log**命令显示从最近到最远的提交日志，我们可以看到3次提交，最近的一次是append GPL，上一次是add distributed，最早的一次是wrote a readme file。

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：

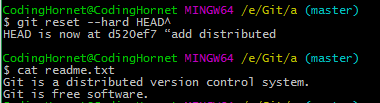


**Tips：**你看到的一大串类似9c2a15b……756911e的是commit id（版本号），和SVN不一样，Git的commit id不是1，2，3……递增的数字，而是一个SHA1计算出来的一个非常大的数字，用十六进制表示，而且你看到的commit id和我的肯定不一样，以你自己的为准。为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

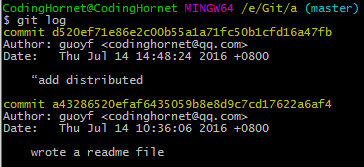
### git reset回退

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交9c2a15b……756911e（注意我的提交ID和你的肯定不一样），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。

现在，我们要把当前版本“append GPL”回退到上一个版本“add distributed”，就可以使用git reset命令：

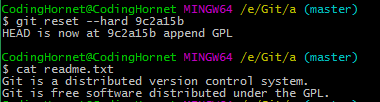


然我们用git log再看看现在版本库的状态：



最新的那个版本append GPL已经看不到了！好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，怎么办？

只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是3628164...，于是就可以指定回到未来的某个版本：



版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是把HEAD从指向append GPL：



改为指向add distributed：

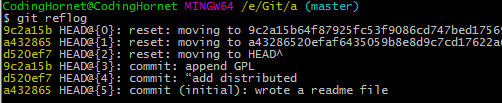


然后顺便把工作区的文件更新了。所以你让HEAD指向哪个版本号，你就把当前版本定位在哪。

### git reflog

现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版本怎么办？找不到新版本的commit id怎么办？

在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add distributed版本时，再想恢复到append GPL，就必须找到append GPL的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

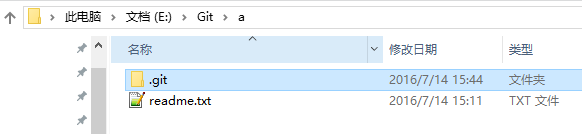


第4行显示append GPL的commit id是9c2a0ef

## 工作区和暂存区和版本库

### 工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的a文件夹就是一个工作区：



### 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。



### 暂存区（Stage）

前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用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；

第二步是用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。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

现在对readme.txt做个修改，比如加上一行内容：

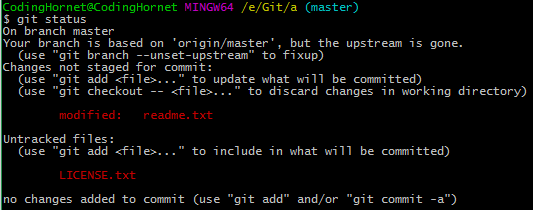
Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

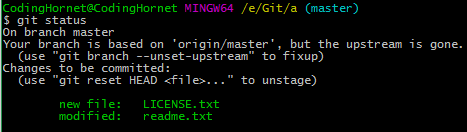
然后，在工作区新增一个LICENSE文本文件（内容随便写）。

先用git status查看一下状态：



Git非常清楚地告诉我们，readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked。

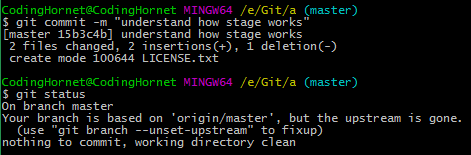
现在，使用两次命令git add，把readme.txt和LICENSE都添加后，用git status再查看一下：



现在，暂存区的状态就变成这样了：



所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。



一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

现在版本库变成了这样，暂存区就没有任何内容了：



## 管理修改

为什么Git比其他版本控制系统设计得优秀，因为Git跟踪并管理的是修改，而非文件。

什么是修改？比如你新增了一行，这就是一个修改，删除了一行，也是一个修改，更改了某些字符，也是一个修改，删了一些又加了一些，也是一个修改，甚至创建一个新文件，也算一个修改。

为什么说Git管理的是修改，而不是文件呢？我们还是做实验。第一步，对readme.txt做一个修改，比如加一行内容：

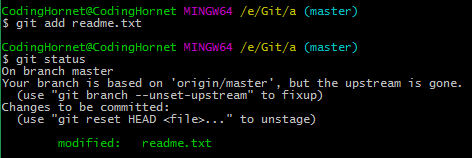
Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes.

然后，添加：



然后，再修改readme.txt：

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

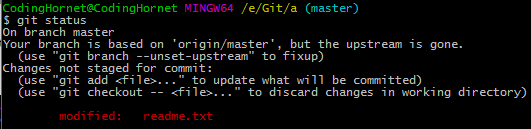
Git has a mutable index called stage.

Git tracks changes of files.

提交：

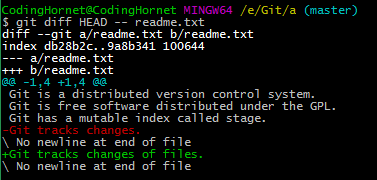


提交后，再看看状态：



当你用git add命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

提交后，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：



可见，第二次修改确实没有被提交。

所以，每次修改，如果不add到暂存区，那就不会加入到commit中。

## 撤销修改

### 场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。你可以发现，Git会告诉你，git checkout -- file可以丢弃工作区的修改：

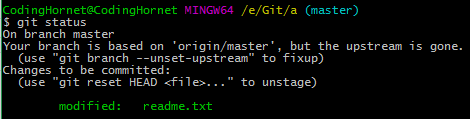
命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

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；

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。

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

### 场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

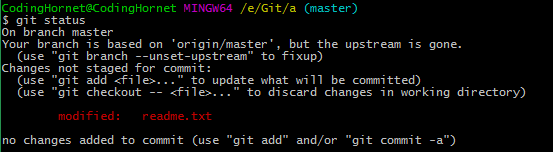


Git同样告诉我们，用命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区：



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。

再用git status查看一下，现在暂存区是干净的，工作区有修改：



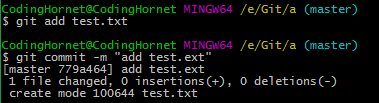
重复场景1的操作



### 场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](#_版本回退)一节，不过前提是没有推送到远程库。

## 删除文件

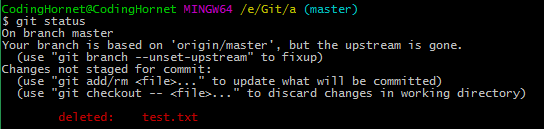
在Git中，删除也是一个修改操作，我们实战一下，先添加一个新文件test.txt到Git并且提交：



一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用rm命令删了：



这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了：

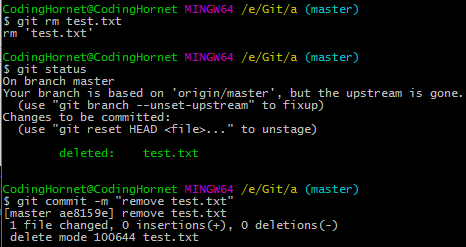


现在你有两个选择，一是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：



git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

另一种情况是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：



现在，文件就从版本库中被删除了。

## 远程仓库

### 添加远程库

要添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用,命令格式如下：

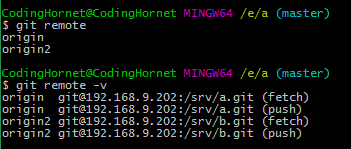
**git remote add [shortname] [url]**



### 查看当前的远程库

要查看当前配置有哪些远程仓库，可以用命令：

**git remote**



执行时加上 -v 参数，你还可以看到每个别名的实际链接地址

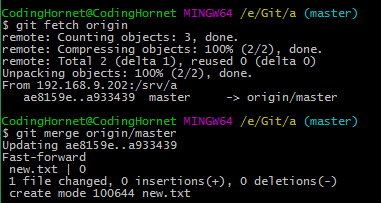
### 提取远程仓库

Git 有两个命令用来提取远程仓库的更新。

* 从远程仓库下载新分支与数据：

**git fetch**

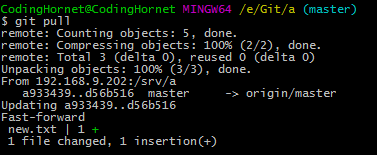
该命令执行完后需要执行git merge 远程分支到你所在的分支。



* 从远端仓库提取数据并尝试合并到当前分支：

**git pull**

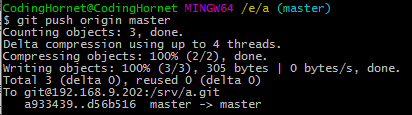
该命令就是在执行 git fetch 之后紧接着执行 git merge 远程分支到你所在的任意分支。



假设你配置好了一个远程仓库，并且你想要提取更新的数据，你可以首先执行 **git fetch [alias]** 告诉 Git 去获取它有你没有的数据，然后你可以执行 **git merge [alias]/[branch]** 以将服务器上的任何更新（假设有人这时候推送到服务器了）合并到你的当前分支。

### 推送到远程仓库

**git push [alias] [branch]**



### 删除远程仓库

删除远程仓库你可以使用命令：

**git remote rm [alias]**

