git教程(3) 时光机穿梭

一 版本回退
1 像这样，你不断对文件进行修改，然后不断提交到版本库里。就好比玩游戏，不断的去存档。打boss之前，手动存档，打不过boss,可以选择从最近的地方重新开始！！
git一样，每当你觉得文件修改到一定程度后【生成快照】，就可以“保存一个快照”【即 git 所讲的 commit。每一次 commit 后会形成“新的版本”！！】.
一旦你把文件该乱了或者误删了文件，还可以从最近的一个 commit恢复，然后继续工作，而不是把几个月的工作成果全部丢失。

```
$ git add readme.txt
$ git commit -m "append GPL"
[master 1094adb] append GPL
 1 file changed, 1 insertion(+), 1 deletion(-)
```

2 回顾一下 readme.txt 文件一共有多少个版本被提交到 git仓库里了：

版本1：wrote a readme file

```
Git is a version control system.
Git is free software.
```

版本2：add distributed

```
Git is a distributed version control system.
Git is free software.
```

版本3：append GPL

```
Git is a distributed version control system.
Git is free software distributed under the GPL.
```

当然实际工作中，自己不可能清晰记得一个几千行的文件都修改了什么内容，所以这是版本控制系统的大作用体现出来了。 ---> git log【时间是倒序排列，最新的永远在前面！！】

```
$ git log
commit 1094adb7b9b3807259d8cb349e7df1d4d6477073 (HEAD -> master)
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 21:06:15 2018 +0800

    append GPL

commit e475afc93c209a690c39c13a46716e8fa000c366
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 21:03:36 2018 +0800

    add distributed

commit eaadf4e385e865d25c48e7ca9c8395c3f7dfaef0
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Fri May 18 20:59:18 2018 +0800

    wrote a readme file
```

注意：如果线上面输出的信息太多，可以 带上参数 --pretty=oneline【按一行一行排列使其变得好看！！！】 ---> git log --pretty=oneline

3 怎么回退版本？？
首先必须知道当前是哪个版本。git中，HEAD表示当前版本，即 最新的提交！！ 每一版本就在HEAD后后面加上^，如上一个版本 HEAD^、上上版本 HEAD^^【与HEAD～2等价】。
回退版本使用 git reset 命令：
git reset --hard【作用？？ 强制？？】 HEAD^（回退前100版本 HEAD~100）

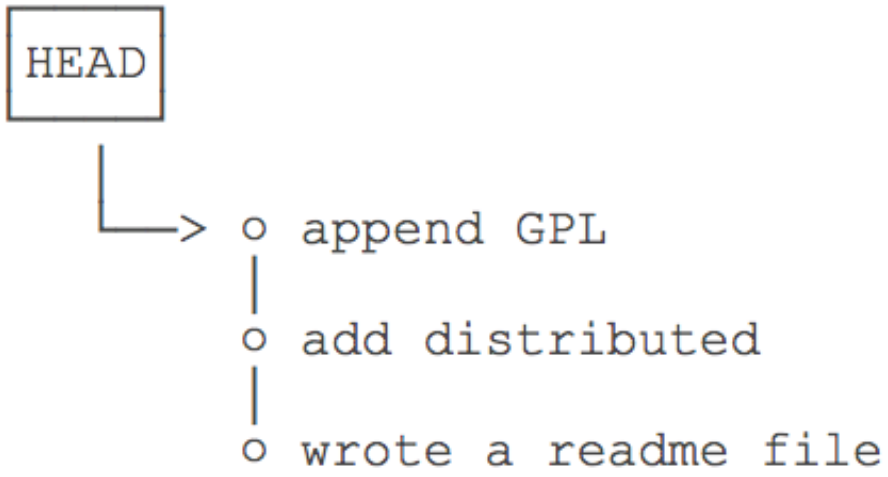
```
$ git reset --hard HEAD^
HEAD is now at e475afc add distributed
```

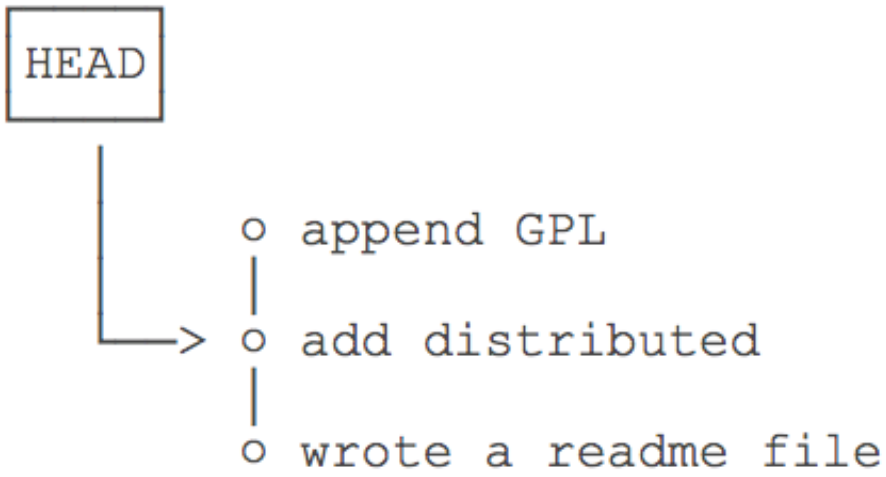
4 那回退了版本，如何恢复到原来最新的哪个版本呢？？【也是 git reset --hard， 通过 传入相应的commit id 值】

```
$ git reset --hard 1094a
HEAD is now at 83b0afe append GPL
```

注意：版本号不必写全，git会自动去找，当然写的太少，可能git就找不清了【匹配上了多个版本号！！！】

5 版本回退之快 —— 核心：指针！！
git的版本回退速度很快，因为 git在内部有个指向当前版本的 HEAD指针，当进行版本回退时，git仅仅把 HEAD 指向改了 —— 从 append GPL 改指向了 add distributed 【所以HEAD后面多少个 ^ 表示 HEAD指针往下移动多少次、版本回退多少次！！ 然后顺便把 工作区 的文件更新了！！】。





6 回退了，又想回到新版本，但找不到 commit_id 怎么办？？【git reflog】
git reflog用来记录你的每一次命令。

```
$ git reflog
e475afc HEAD@{1}: reset: moving to HEAD^
1094adb (HEAD -> master) HEAD@{2}: commit: append GPL
e475afc HEAD@{3}: commit: add distributed
eaadf4e HEAD@{4}: commit (initial): wrote a readme file
```

7 小结

7.1 HEAD就是一个指针，指向当前“工作区”【？？应该是吧？】的版本，git允许我们在各个版本之间来回穿梭。 ---> git reset --hard commit_id。

7.2 穿梭到历史，用 git log 查看提交历史，以便 回退到某个提交的历史版本！

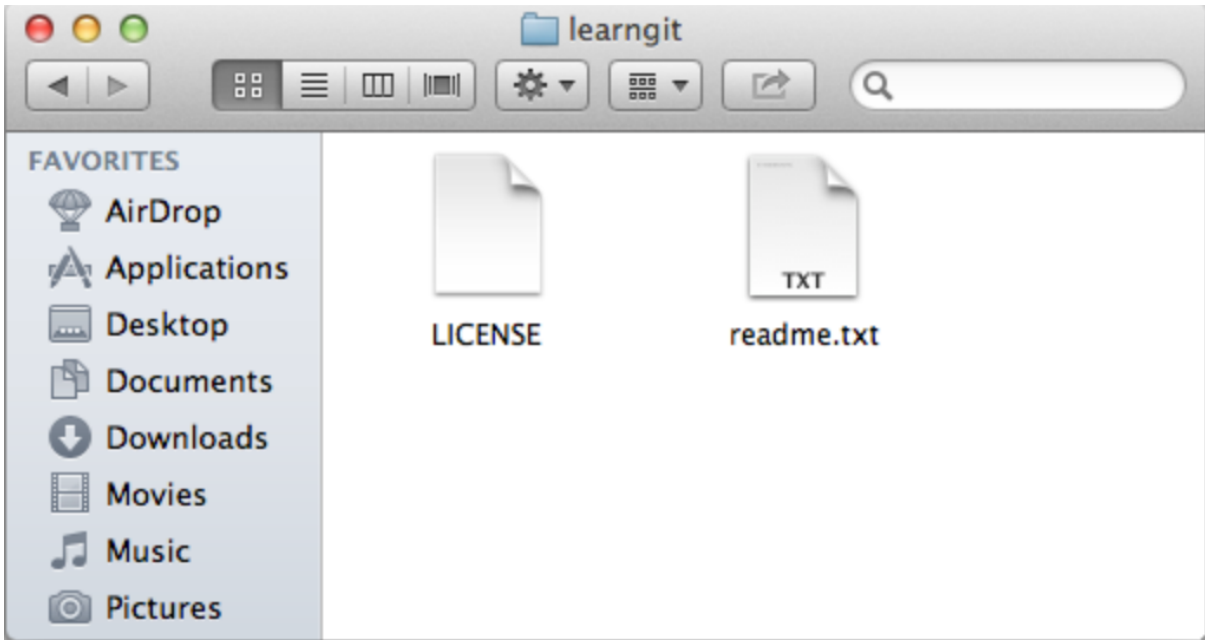
7.3 重返未来。用 git reflog 查看命令历史，获取到相应的 commit——id 以便确定要回到未来【“较新的”】的哪个版本！！

二 工作区 和 暂存区

1 git【多了暂存区的概念】 和其他版本控制系统如 SVN的不同之处就是有 暂存区【注意区分 工作区的 概念】的概念。

2 工作区【就是你在电脑能看到的目录。】

如下面的 learngit文件夹就是一个工作区：



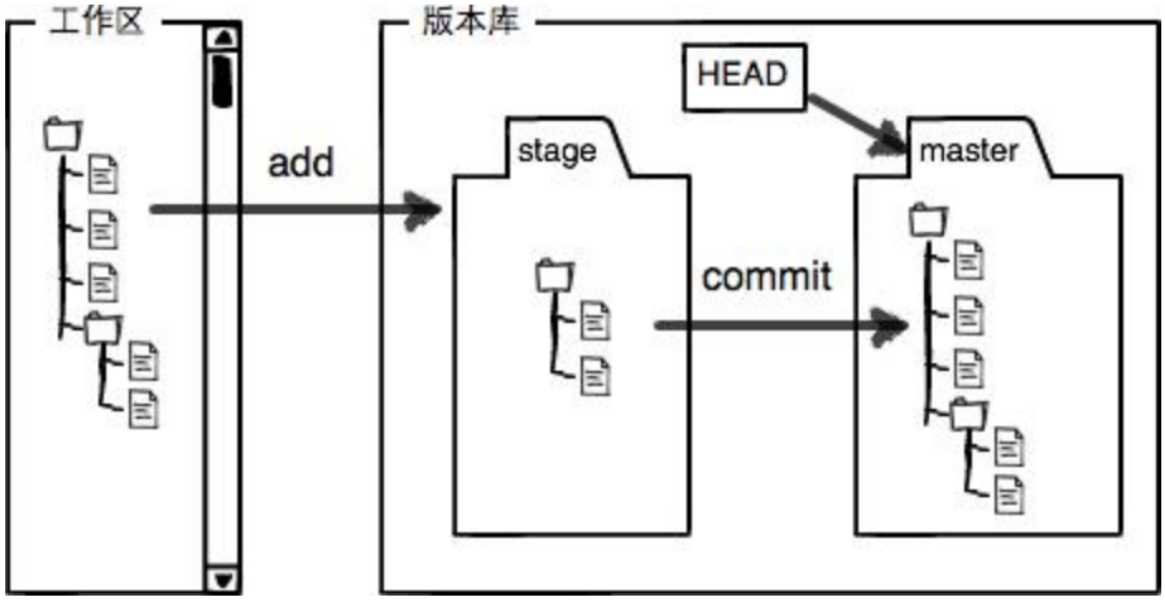
3 版本库【.git隐藏目录 就是 git版本库】：

工作区有一个隐藏的目录 .git ， 这个不算工作区，而是称为 git版本库。

3.2 git的版本库存了 很多东西【stage暂存区 + master等分支】，其中最重要的就是

stage（或叫index）的暂存区，还有 git为我们自动创建的第一个分支matser，以及指向 master 的一个指针 HEAD。

3.3 工作区内的内容修改并 git add . 操作后，会把内容存到暂存区【stage】；
暂存区stage通过 git commit 去改变 HEAD指针的指向？？？！。



故：
第一步是用 git add 把文件添加进去。实际就是把文件的修改 添加到 暂存区【“这样一来，工作区就清空了吧？？！”是的】。
第二步 用 git commit 提交更改，实际就是把 暂存区 的所有内容提交到当前分支。

4 小结
暂存区【位于版本库中】，工作区【“除去.git目录的文件夹”】的修改 --git add--> 版本库【.git目录下】中暂存区 --git commit--> 暂存取的东西提交到分支中【移动 HEAD指针的指向】。

三 管理修改
1 为啥git【跟踪并管理修改，而不是文件！！！】比其他版本控制系统设计得更优秀，因为git跟踪并管理的是修改， 而不是文件！！

2 为啥说 git管理的是修改，而不是文件？？
2.1 对 readme.txt 文件进行修改，如添加一行代码

```
$ cat 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.
```

2.2 然后添加至暂存区stage，git add

```
$ git add readme.txt
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   readme.txt
#
```

2.3 然后继续修改 readme.txt

```
$ cat 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.
```

2.4 最后的一次的修改不进行 git add 操作！！直接 git commit 前前一次的修改！！

```
$ git commit -m "git tracks changes"
[master 519219b] git tracks changes
 1 file changed, 1 insertion(+)
```

2.5 提交之后，再看看状态 git status【发现提示 readme.txt 文件被修改了，但没有被 git add！！ 即 第二次修改没有被提交！！】

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

2.6 回顾一下刚才的操作

第一次修改 --> git add --> 第二次修改【未进行git add】 --> git commit。

所以 git add 使得在工作区的第一次修改被添加至暂存区了、准备提交，但是第二次修改没有被添加到暂存区，所以 git commit 只是把第一次修改提交了、第二次修改不会被提交！！

注意：commit提交之后，可以进行对比 ---> 通过 git diff HEAD【版本库的最新版本】 -- readme.txt 查看 工作区 和 版本库里买你最新版本的区别：

```
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
index 76d770f..a9c5755 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
 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.
+Git tracks changes of files.
```

2.7 以上的“解决方案”：

那怎么提交第二次修改呢？你可以继续 `git add` 再 `git commit`，也可以别着急提交第一次修改，先 `git add` 第二次修改，再 `git commit`，就相当于把两次修改合并后一块提交了：

第一次修改 -> `git add` -> 第二次修改 -> `git add` -> `git commit`

3 小结：

每次修改，如果不用 git add 到暂存区，那就不会加入到 commit 中！！ 【一般流程，在工作区对文件进行修改 --> git add . 将修改的内容添加至 暂存区stage --> git commit 将暂存区所保存的修改内容提交至 “版本库”，HEAD指针进行移动操作！！】

四 撤销修改

1

1.1 不小心犯错了，如何撤销所做的修改？？ 【最后一行改错了！！】

```
$ cat 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.
My stupid boss still prefers SVN.
```

1.2 使用 git status查看一下当前状态

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

1.3 我们会发现有用的提示 —— git checkout -- file【如果 git checkout -- readme.txt。如果没有 -- 就变成了切换分支的操作了！！】，去放弃对工作区的修改！！

1.4 撤销修改有2种情况：

自修改后还 没放到暂存区【恢复成版本库一样的状态！！】，撤销修改会使其回到和 版本库 一样的状态！

已经通过git add添加至暂存区，然后又作了修改，那撤销就是回到最近一次暂存区的状态！！

总之，git checkout -- file 撤销修改，就是使得某文件回到最近一次 commit【最近的一次版本库】 或 add【最近一次暂存区】 的状态！！

注意：撤销操作别忘了 -- ，没有这个就变成了 切换分支操作了！！ 【git checkout branch_name 和 git checkout -- file_name】

2 写错了还 git add到了暂存区，怎么办？？

2.1 庆幸的是只进行了 git add操作，没有进行 commit操作

```
$ cat 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.
My stupid boss still prefers SVN.

$ git add readme.txt
```

2.2 git status会发现修改只是放进了暂存区，还没有修改！！ 【此时 工作区也算是 “干净的”】

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   readme.txt
```

2.3 git也输出了有用的提示。git reset HEAD <file> 可以把暂存取的修改撤销掉（unstage），重新放回工作区！！

```
$ git reset HEAD readme.txt
Unstaged changes after reset:
M   readme.txt
```

2.4 再用git status查看一下。发现 暂存区确实干净了、但 工作区变成了有修改的状态！！

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   readme.txt
```

2.5 按如何也让工作区变得清净呢？？【提示：撤销修改操作 --> git checkout -- readme.txt】

```
$ git checkout -- readme.txt

$ git status
On branch master
nothing to commit, working tree clean
```

2.6 git reset很强大【可以回退版本 + （暂存区的修改 --> 工作区）把暂存区的修改回退到工作区】

3 假设不但改错了东西，还从暂存区提交到了版本库 --> 回退到上一个版本【本质是使HEAD指针发生改变】 --> git reset HEAD^^（等价于 git reset HEAD~2） 。不过这个是有条件的 —— 还没有把自己的版的版本库推送到远程【即 还没有 git push 操作】

4 小结【三种情况的撤销，程度“越来越大”】
4.1 当改乱了工作区某个文件的内容，想直接丢弃工作区的修改 --> git checkout -- file_name【别漏了 -- ，不然变成了切换分支！！！】

4.2 不但改乱了工作区的某个文件的内容，还添加至了暂存区，此时又想放弃修改，分 2步：
第一、git reset HEAD <file_name> 就回到了 4.1 ，接着就是 git checkout -- file_name 把从暂存区回退到工作区的修改内容给撤销掉！！

4.3 前提是还没推送至远程库。已经提交了不适合的修改到 版本库时，想撤销本次提交【“为啥叫提交，push 至远程的 提交？？！”】 ---> git reset HEAD^^（等同于 git reset HEAD~2）

五 删除文件
1 git中，删除也是一个修改操作。

1.1 先添加一个新的文件 test.txt 到git并进行提交操作！！

```
$ git add test.txt

$ git commit -m "add test.txt"
[master b84166e] add test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

1.2 一般情况下，是通过 rm命令去删除文件。

```
$ rm test.txt
```

这个时候，git知道你删除了文件，因此工作区和版本库【每一次commit会形成一个版本！！】就不一致了。

1.3 git status 去查看当前状态【rm test.txt之后】

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

1.4 从提示信息中可知，我们有 2种选择。若确实需要从版本库中删除该文件，就用 命令 git rm 删除掉 且在进行 commit操作！！

```
$ git rm test.txt
rm 'test.txt'

$ git commit -m "remove test.txt"
[master d46f35e] remove test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

这样，文件就从版本库中删除了！！！

如果是不小心 rm错文件了，不用担心、因为版本库还保存着呢！！ 可以很轻松的将误删文件 恢复回来 —— git checkout -- test.txt【跟之前撤销工作区的修改一样？！！】

```
$ git checkout -- test.txt
```

注意：git checkout 其实就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 或 删除， 都可以一键还原！！
【对】但是从来 没被添加至版本库 就被删除的文件，是无法恢复的！！

2 小结
命令 git rm file_name 用于删除一个文件。
如果一个文件已经 被提交至版本库，那么你永远不用担心误删！！！
但要小心，你只能恢复文件到最新版本，你会丢失 最近一次提交后你所做的修改！！

完