时空机穿梭

2021年4月19日 10:41

我们已经成功地添加并提交了一个readme.txt文件，现在，是时候继续工作了，于是，我们继续修改readme.txt文件，改成如下内容：

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

现在，运行git status命令看看结果：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

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

git status命令可以让我们时刻掌握仓库当前的状态，上面的命令输出告诉我们，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。比如你休假两周从国外回来，第一天上班时，已经记不清上次怎么修改的readme.txt，所以，需要用git diff这个命令看看：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index d22be28..3a94755 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1 +1,2 @@
-Git is a version control system.
+Git is a version control system.
+Git is free software.
\ No newline at end of file
```

之前的

修改后的

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。

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，提交修改和提交新文件是一样的两步。

第一步是git add：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git add readme.txt
```

同样没有任何输出。在执行第二步git commit之前，我们再运行git status看看当前仓库的状态：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.txt
```

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：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git commit -m "add a new line git is free software"
[master a8130c4] add a new line git is free software
 1 file changed, 2 insertions(+), 1 deletion(-)
```

提交后，我们再用git status命令看看仓库的当前状态：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Git告诉我们当前没有需要提交的修改，而且，工作目录是干净（working tree clean）的。

>

版本回退

2021年4月19日　16:26

你不断对文件进行修改，然后不断提交修改到版本库里，就好比玩RPG游戏时，每通过一关就会自动把游戏状态存盘，如果某一关没过去，你还可以选择读取前一关的状态。有些时候，在打Boss之前，你会手动存盘，以便万一打Boss失败了，可以从最近的地方重新开始。

Git也是一样，每当你觉得文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，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。

在实际工作中，我们脑子里不能记得一个几千行的文件每次都改了什么内容。版本控制系统肯定有某个命令可以告诉我们历史记录，在Git中，我们用git log命令查看：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git log
commit 7d8b7650531ad5efd0d77c1bfc00a68e508830fa (HEAD -> master)
Author: juliazhu <1340772095@qq.com>
Date:   Mon Apr 19 16:31:05 2021 +0800

    append GPL

commit a8130c416283c07e01898cb7becf6a463f2aa293
Author: juliazhu <1340772095@qq.com>
Date:   Mon Apr 19 16:22:55 2021 +0800

    add a new line git is free software

commit 00190f00c86c8dafb142bcb6f56e7d5b5924620f
Author: juliazhu <1340772095@qq.com>
Date:   Mon Apr 19 15:58:00 2021 +0800

    wrote a readme file
```

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 a new line git is free software，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：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git log --pretty=oneline
7d8b7650531ad5efd0d77c1bfc00a68e508830fa (HEAD -> master) append GPL
a8130c416283c07e01898cb7becf6a463f2aa293 add a new line git is free software
00190f00c86c8dafb142bcb6f56e7d5b5924620f wrote a readme file
```

版本号

需要友情提示的是，你看到的一大串类似7d8b76...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.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。

每提交一个新版本，实际上Git就会把它们自动串成一条时间线。如果使用可视化工具查看Git历史，就可以更清楚地看到提交历史的时间线：

现在我们启动时光穿梭机，准备把readme.txt回退到上一个版本，也就是add a new line git is free software的那个版本，怎么做呢？

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7d8b76，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 a new line git is free software，就可以使用git reset命令：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git reset --hard HEAD^
HEAD is now at a8130c4 add a new line git is free software
```

看看readme.txt的内容是不是版本add a new line git is free software：

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ cat readme.txt
Git is a version control system.
Git is free software.
```

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git log
commit a8130c416283c07e01898cb7becf6a463f2aa293 (HEAD -> master)
Author: juliazhu <1340772095@qq.com>
Date:   Mon Apr 19 16:22:55 2021 +0800

    add a new line git is free software

commit 00190f00c86c8dafb142bcb6f56e7d5b5924620f
Author: juliazhu <1340772095@qq.com>
Date:   Mon Apr 19 15:58:00 2021 +0800
```

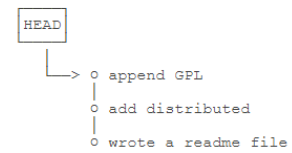
最新的那个版本append GPL已经看不到了！好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，肿么办？

办法其实还是有的，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7d8b76...，于是就可以指定回到未来的某个版本：

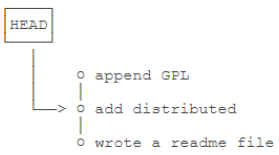
```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git reset --hard 7d8b7
HEAD is now at 7d8b765 append GPL
```

==========

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。

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：

```
$ 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
```

==========

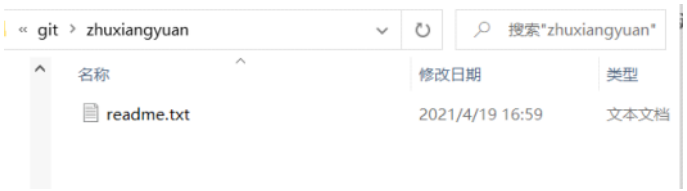
1. HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit_id。
2. 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
3. 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

工作区和暂存区

2021年4月19日 17:07

工作区（Working Directory）

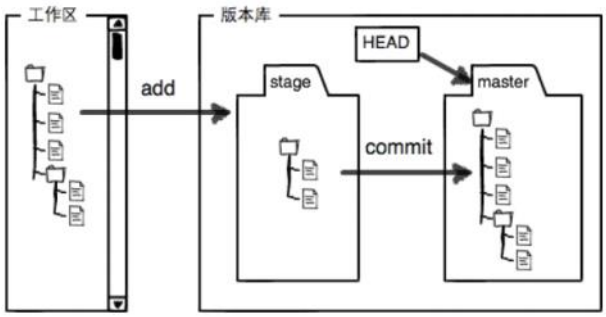
就是你在电脑里能看到的目录，比如我的zhuxiangyuan文件夹就是一个工作区：



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。



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。

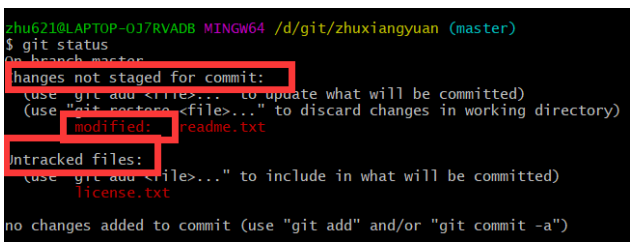
你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

==========

先对readme.txt做个修改，比如加上一行内容：

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：

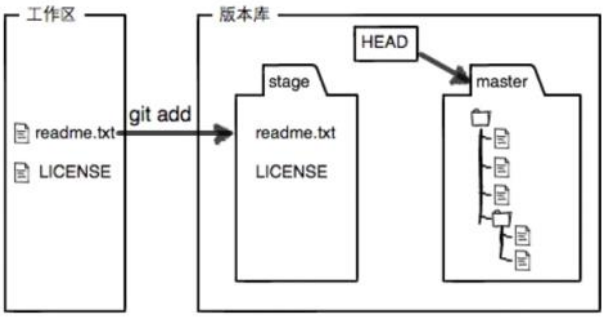
```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git add readme.txt

zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git add licsnse.txt
fatal: pathspec 'licsnse.txt' did not match any files

zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git add license.txt

zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   license.txt
        modified:   readme.txt
```

现在，暂存区的状态就变成这样了：



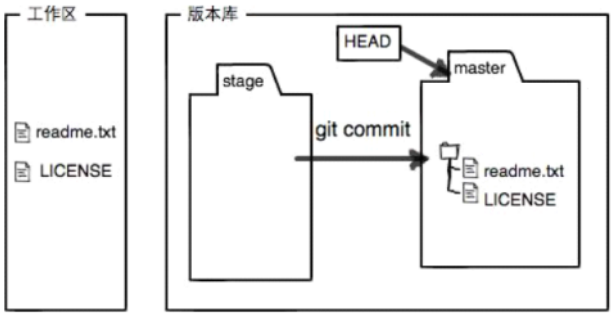
所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git commit -m "understand how stage works"
[master d3099ba] understand how stage works
 2 files changed, 2 insertions(+)
 create mode 100644 license.txt

zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
nothing to commit, working tree clean
```

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

2021年4月19日 17:30

现在，假定你已经完全掌握了暂存区的概念。下面，我们要讨论的就是，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。

你会问，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？

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：

```
$ 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.
```

然后，添加：

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

然后，再修改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.
```

提交：

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

提交后，再看看状态：

```
$ 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")
```

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

你看，我们前面讲了，Git管理的是修改，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。

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

==========

每次修改，如果不用git add到暂存区，那就不会加入到commit中。

>

撤销修改

2021年4月19日 17:40

第一种情况：还没有被放到暂存区

```
$ 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 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")
```

```
$ git checkout -- 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.
```

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：

1. 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；
2. 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。

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，我们在后面的分支管理中会再次遇到git checkout命令。

==========

第二种情况：已经被放到暂存区，又做了修改

```
$ 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
```

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

    modified:   readme.txt
```

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

第一步，放回到工作区

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。

```
$ 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
```

```
$ git checkout -- readme.txt

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

第二步，在工作区撤回

删除文件

2021年4月19日 20:26

在Git中，删除也是一个修改操作

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git add test.txt

zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git commit -m"add test.txt"
[master 25d48a6] add test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

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：

```
$ rm test.txt
```

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

第一种：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：文件就从版本库中被删除了。

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        license.txt

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

```
zhu621@LAPTOP-OJ7RVADB MINGW64 /d/git/zhuxiangyuan (master)
$ git commit -m"remove test.txt"
[master 0910ca0] remove test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

==========

第二种：删错了

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 -- test.txt
```

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”。

==========

注意：从来没有被添加到版本库就被删除的文件，是无法恢复的！

命令git rm用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容。