《软件工程》实训指导书——Git 使用 3

陈晓华 qq：78976932 微信号：chen-jeo 日期：2017.2.20

一、实验目的

掌握版本回退、管理修改、撤销修改、删除文件。

二、实验内容

在工作区和暂存区中管理文件。

三、实验步骤

版本回退、管理修改、撤销修改、删除文件。

四、参考实验过程

1、版本回退。

再练习一次修改 readme.txt 文件如下：

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

然后尝试提交：

```
$ git add readme.txt
$ git commit -m "append GPL"
[master 3628164] append GPL
 1 file changed, 1 insertion(+), 1 deletion(-)
```

像这样，你不断对文件进行修改，然后不断提交修改到版本库里。每当你觉得文件修改到一定程度的时候，Git 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。

现在，我们回顾一下 readme.txt 文件一共有几个版本被提交到 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.
```

当然了，在实际工作中，版本控制系统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：

```
$ git log
commit 3628164fb26d48395383f8f31179f24e0882e1e0
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Tue Aug 20 15:11:49 2013 +0800
    append GPL
commit ea34578d5496d7dd233c827ed32a8cd576c5ee85
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Tue Aug 20 14:53:12 2013 +0800
    add distributed
commit cb926e7ea50ad11b8f9e909c05226233bf755030
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Mon Aug 19 17:51:55 2013 +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 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：

```
$ git log --pretty=oneline
3628164fb26d48395383f8f31179f24e0882e1e0 append GPL
ea34578d5496d7dd233c827ed32a8cd576c5ee85 add distributed
cb926e7ea50ad11b8f9e909c05226233bf755030 wrote a readme file
```

需要友情提示的是，你看到的一大串类似 3628164...882e1e0 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。

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：

好了，准备把 readme.txt 回退到上一个版本，也就是“add distributed”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 3628164...882e1e0（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：

```
$ git reset --hard HEAD^
HEAD is now at ea34578 add distributed
```

--hard 参数有啥意义？这个后面再讲，现在你先放心使用。

看看 readme.txt 的内容是不是版本 add distributed：

```
$ cat readme.txt
Git is a distributed version control system.
Git is free software.
```

还可以继续回退到上一个版本 wrote a readme file，不过且慢，然我们用 git log 再看看现在版本库的状态：

```
$ git log
commit ea34578d5496d7dd233c827ed32a8cd576c5ee85
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Tue Aug 20 14:53:12 2013 +0800
    add distributed
commit cb926e7ea50ad11b8f9e909c05226233bf755030
Author: Michael Liao <askxuefeng@gmail.com>
Date:   Mon Aug 19 17:51:55 2013 +0800
    wrote a readme file
```

最新的那个版本 append GPL 已经看不到了！怎么办？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 3628164...，于是就可以指定回到未来的某个版本：

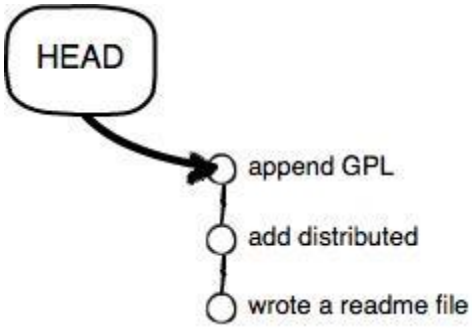
```
$ git reset --hard 3628164
HEAD is now at 3628164 append GPL
```

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。再小心翼翼地看看 readme.txt 的内容：

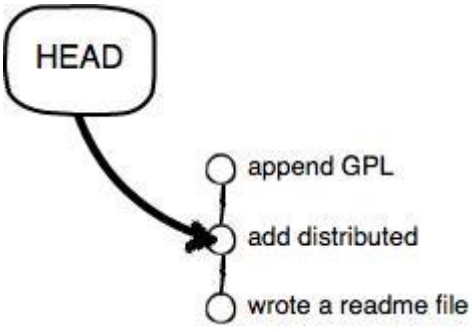
```
$ cat readme.txt
Git is a distributed version control system.
Git is free software distributed under the GPL.
```

果然，又回来了。

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。

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，总是有办法的。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
ea34578 HEAD@{0}: reset: moving to HEAD^
3628164 HEAD@{1}: commit: append GPL
ea34578 HEAD@{2}: commit: add distributed
cb926e7 HEAD@{3}: commit (initial): wrote a readme file
```

终于舒了口气，第二行显示 append GPL 的 commit id 是 3628164。

现在，假设你不但改错了东西，还从暂存区提交到了版本库，怎么办呢？还记得版本回退一节吗？可以回退到上一个版本。不过，这是有条件的，就是你还没有把自己的本地版本库推送到远程。还记得 Git 是分布式版本控制系统吗？我们后面会讲到远程版本库，一旦你把“stupid boss”提交推送到远程版本库，你就真的惨了……

2、管理修改。

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？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 d4f25b6] 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")
```

咦，怎么第二次的修改没有被提交？

别激动，我们回顾一下操作过程：

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。

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：

```
$ 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.
```

可见，第二次修改确实没有被提交。

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

好，现在，把第二次修改提交了。

3、撤销修改。

现在是凌晨两点，你正在赶一份工作报告，你在 readme.txt 中添加了一行：

```
$ 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.
```

在你准备提交前，一杯咖啡起了作用，你猛然发现了“stupid boss”可能会让你丢掉这个月的奖金！既然错误发现得很及时，就可以很容易地纠正它。你可以删掉最后一行，手动把文件恢复到上一个版本的状态。如果用 git status 查看一下：

```
$ 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")
```

你可以发现，Git 会告诉你，git checkout -- file 可以丢弃工作区的修改：

```
$ git checkout -- readme.txt
```

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。

现在，看看 readme.txt 的文件内容：

```
$ 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 of files.
```

文件内容果然复原了。

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。

现在假定是凌晨 3 点，你不但写了一些胡话，还 git add 到暂存区了：

```
$ 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.
$ git add readme.txt
```

庆幸的是，在 commit 之前，你发现了这个问题。用 git status 查看一下，修改只是添加到了暂存区，还没有提交：

```
$ 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
```

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：

```
$ 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")
```

还记得如何丢弃工作区的修改吗？

```
$ git checkout -- readme.txt
$ git status
# On branch master
nothing to commit (working directory clean)
```

4、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：

```
$ git add test.txt
$ git commit -m "add test.txt"
[master 94cdc44] 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：

```
$ 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")
```

现在你有两个选择，一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：

```
$ git rm test.txt
rm 'test.txt'
$ git commit -m "remove test.txt"
[master d17efd8] remove test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt
```

现在，文件就从版本库中被删除了。

另一种情况是删错了，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 -- 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”。

五、思考与总结

1、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`。

2、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。

3、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，不过前提是没有推送到远程库。

4、Git 是如何跟踪修改的？每次修改，如果不 add 到暂存区，那就不会加入到 commit 中。

5、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。