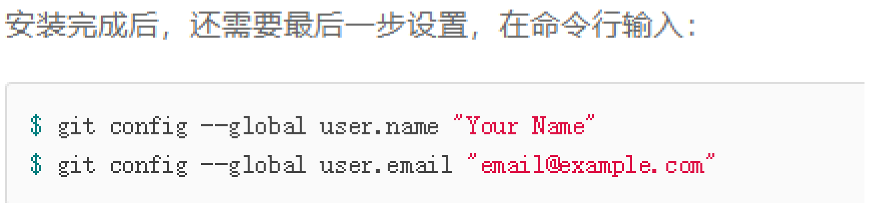
1. windows安装完成 git 软件后需设置名字和email地址



安装完成后，还需要最后一步设置，在命令行输入：

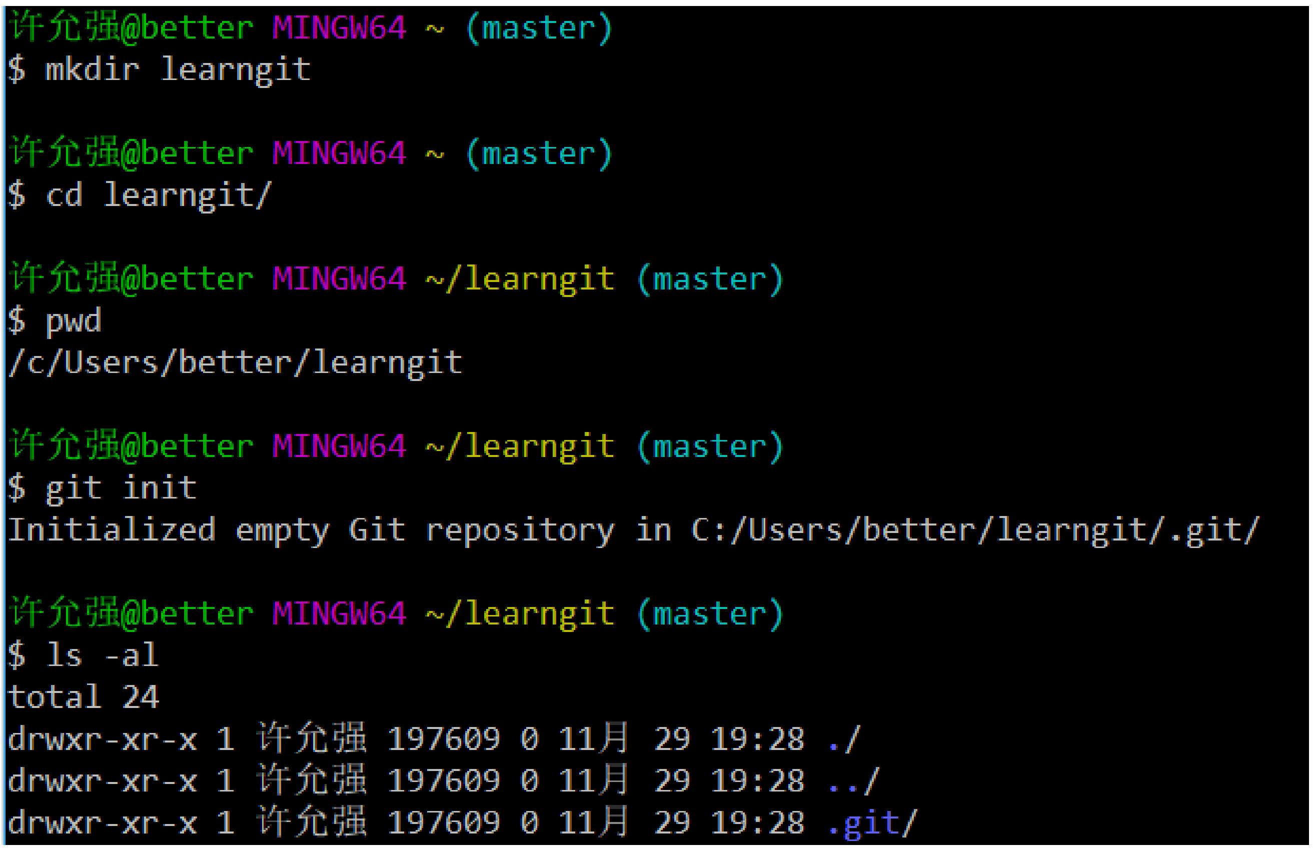
```
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
```

2. 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"。

   通过 `git init` 命令把这个目录变成Git可以管理的仓库

   当前目录下会多一个 `.git` 的目录，这个目录是Git来跟踪管理版本库的，千万不要手动修改这个目录里面的文件

   `.git` 目录默认是隐藏的



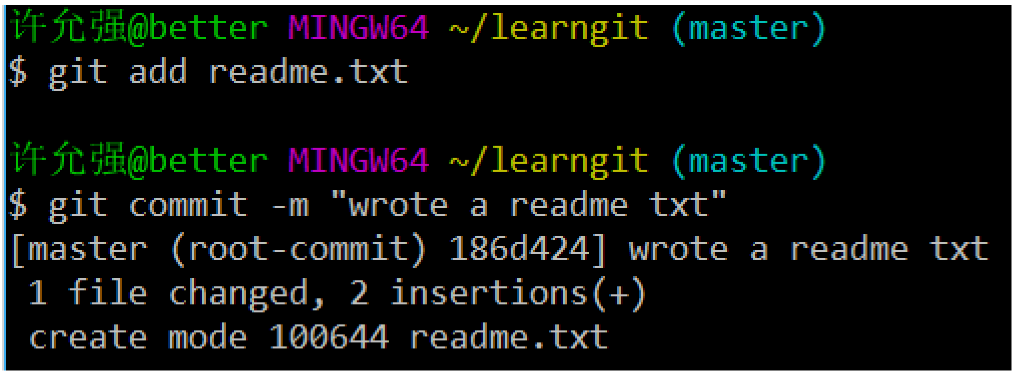
```
许允强@better MINGW64 ~ (master)
$ mkdir learngit

许允强@better MINGW64 ~ (master)
$ cd learngit/

许允强@better MINGW64 ~/learngit (master)
$ pwd
/c/Users/better/learngit

许允强@better MINGW64 ~/learngit (master)
$ git init
Initialized empty Git repository in C:/Users/better/learngit/.git/

许允强@better MINGW64 ~/learngit (master)
$ ls -al
total 24
drwxr-xr-x 1 许允强 197609 0 11月 29 19:28 ./
drwxr-xr-x 1 许允强 197609 0 11月 29 19:28 ../
drwxr-xr-x 1 许允强 197609 0 11月 29 19:28 .git/
```

3. `git add` 把文件添加到仓库

   `git commit` 把文件提交到仓库

   `git commit` 命令， `-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

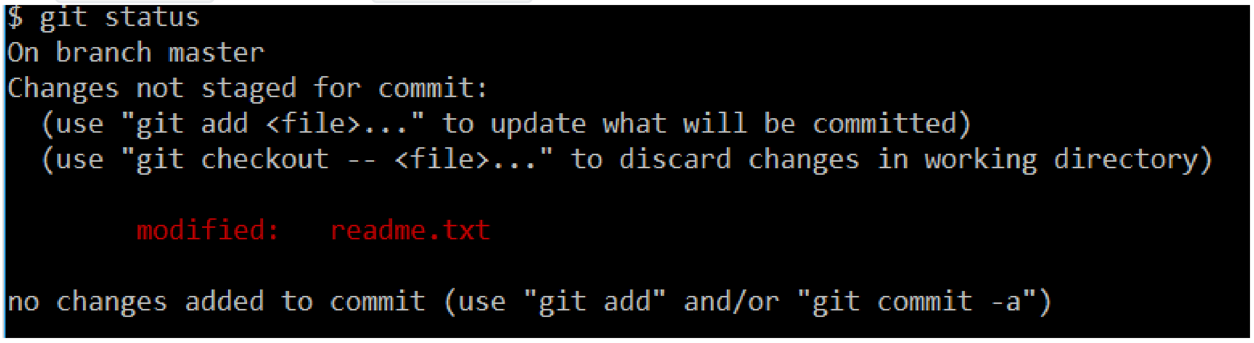
   `commit` 可以一次提交很多文件，所以可以多次 `add` 不同的文件然后一起commit提交



```
许允强@better MINGW64 ~/learngit (master)
$ git add readme.txt

许允强@better MINGW64 ~/learngit (master)
$ git commit -m "wrote a readme txt"
[master (root-commit) 186d424] wrote a readme txt
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt
```

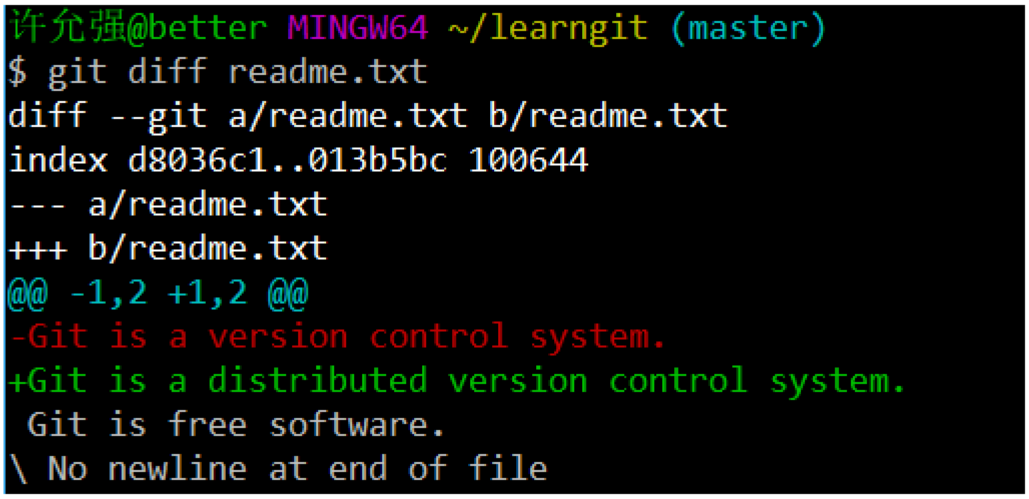
4. `git status` 命令可以让我们时刻掌握仓库当前的状态

   修改 `readme.txt` 文件然后运行 `git status` 看看当前仓库的状态，下面的命令表示文件还没有提交



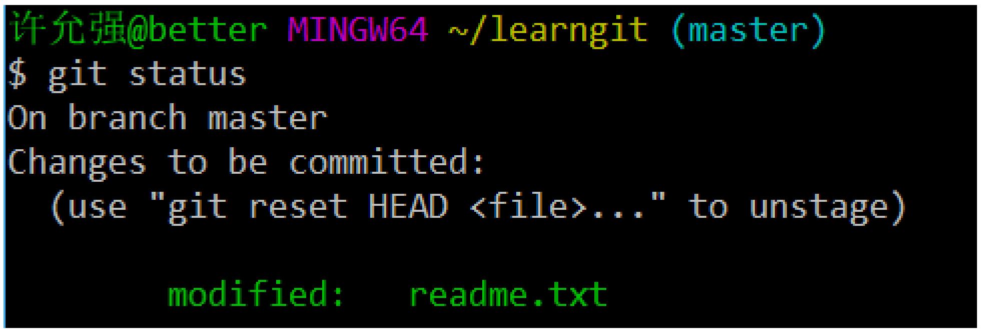
```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

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

   `git diff` 顾名思义就是查看difference，下面的命令输出表示修改后文件第一行添加了一个 `distributed` 单词



```
许允强@better MINGW64 ~/learngit (master)
$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index d8036c1..013b5bc 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,2 @@
-Git is a version control system.
+Git is a distributed version control system.
 Git is free software.
\ No newline at end of file
```

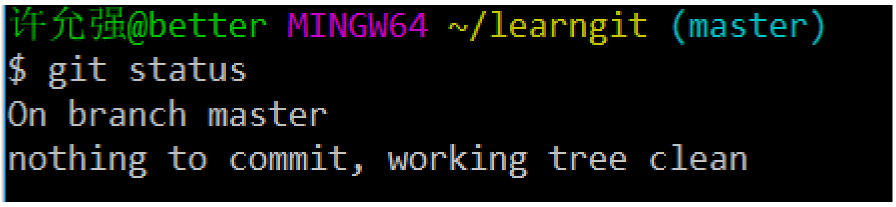
在执行 `git commit` 之前，运行 `git status` 看看当前仓库的状态



```
许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt
```

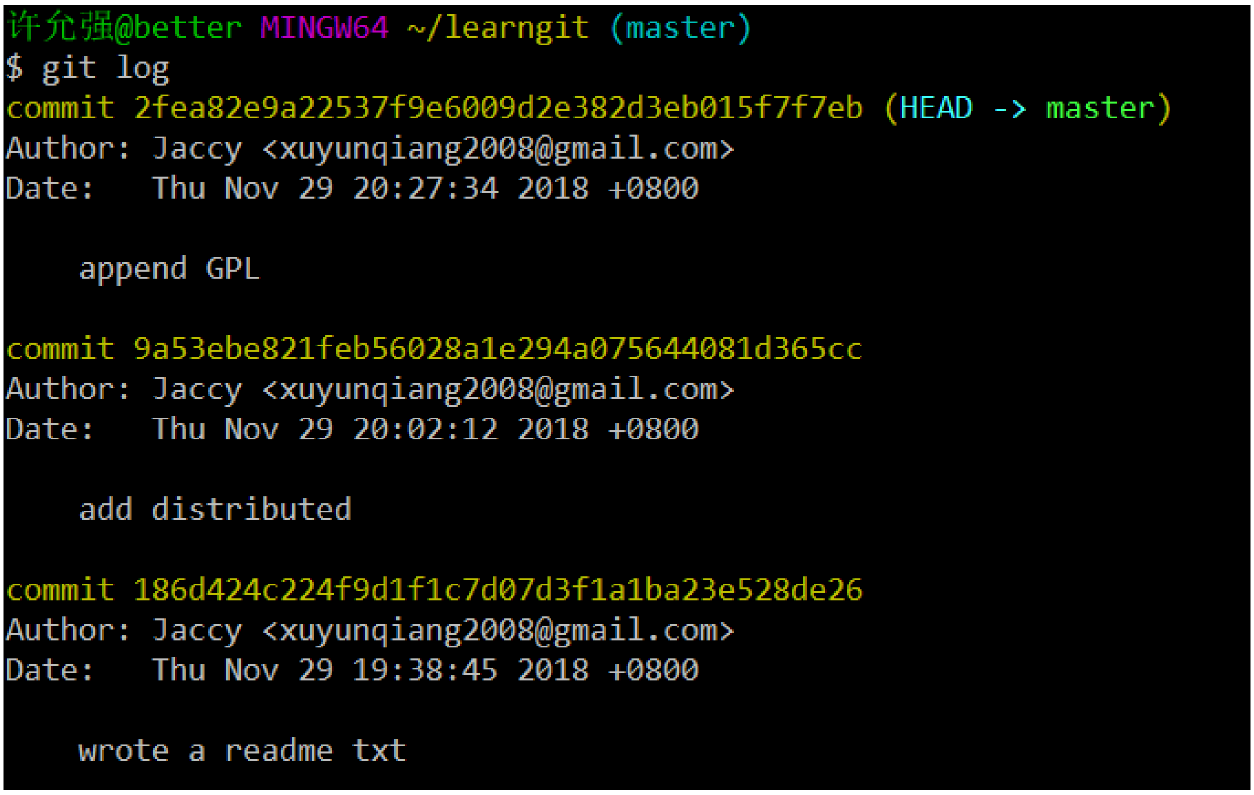
执行 `git commit`，运行 `git status` 看看当前仓库的状态

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



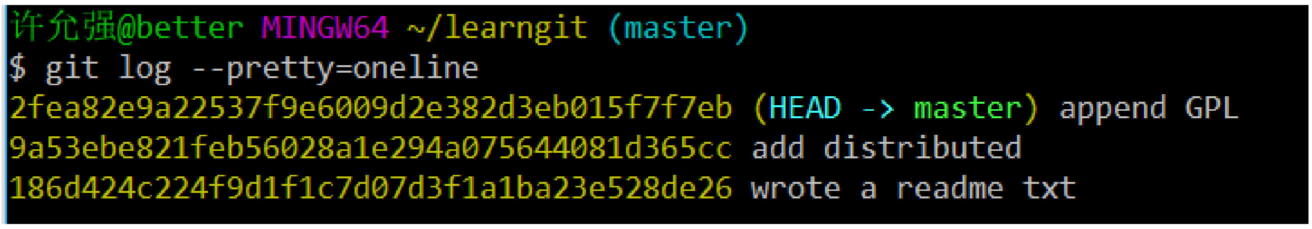
```
许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

5. `git log` 命令查看历史记录，`git log`命令显示从最近到最远的提交日志



```
许允强@better MINGW64 ~/learngit (master)
$ git log
commit 2fea82e9a22537f9e6009d2e382d3eb015f7f7eb (HEAD -> master)
Author: Jaccy <xuyunqiang2008@gmail.com>
Date:   Thu Nov 29 20:27:34 2018 +0800

    append GPL

commit 9a53ebe821feb56028a1e294a075644081d365cc
Author: Jaccy <xuyunqiang2008@gmail.com>
Date:   Thu Nov 29 20:02:12 2018 +0800

    add distributed

commit 186d424c224f9d1f1c7d07d3f1a1ba23e528de26
Author: Jaccy <xuyunqiang2008@gmail.com>
Date:   Thu Nov 29 19:38:45 2018 +0800

    wrote a readme txt
```

`git log` 命令后可以加上`--pretty=oneline`参数，日志信息更简洁

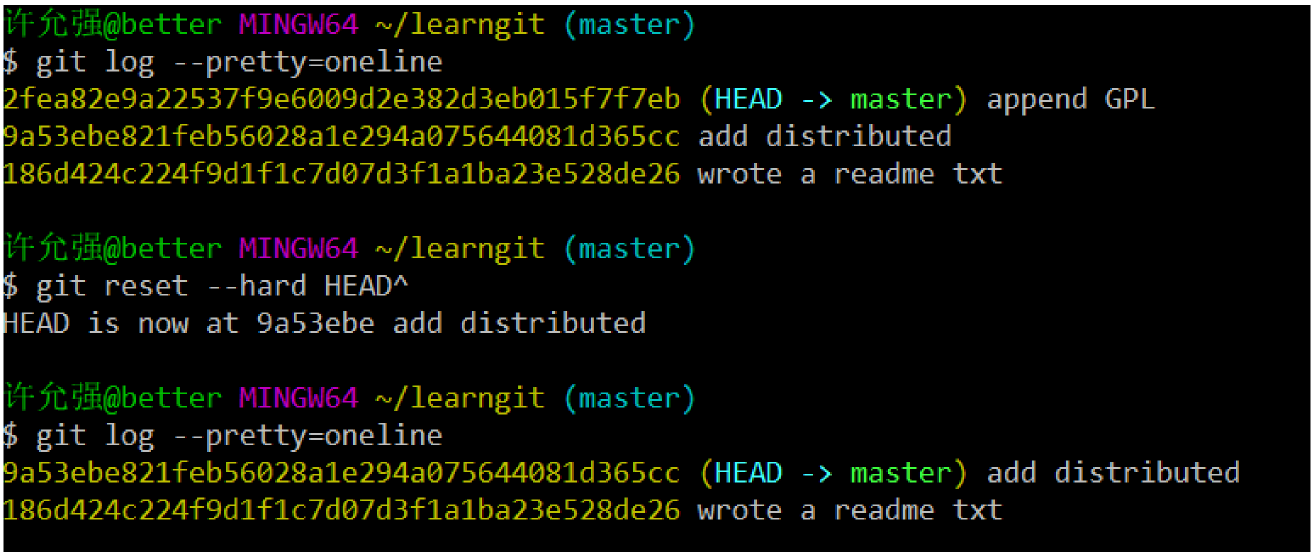


```
许允强@better MINGW64 ~/learngit (master)
$ git log --pretty=oneline
2fea82e9a22537f9e6009d2e382d3eb015f7f7eb (HEAD -> master) append GPL
9a53ebe821feb56028a1e294a075644081d365cc add distributed
186d424c224f9d1f1c7d07d3f1a1ba23e528de26 wrote a readme txt
```

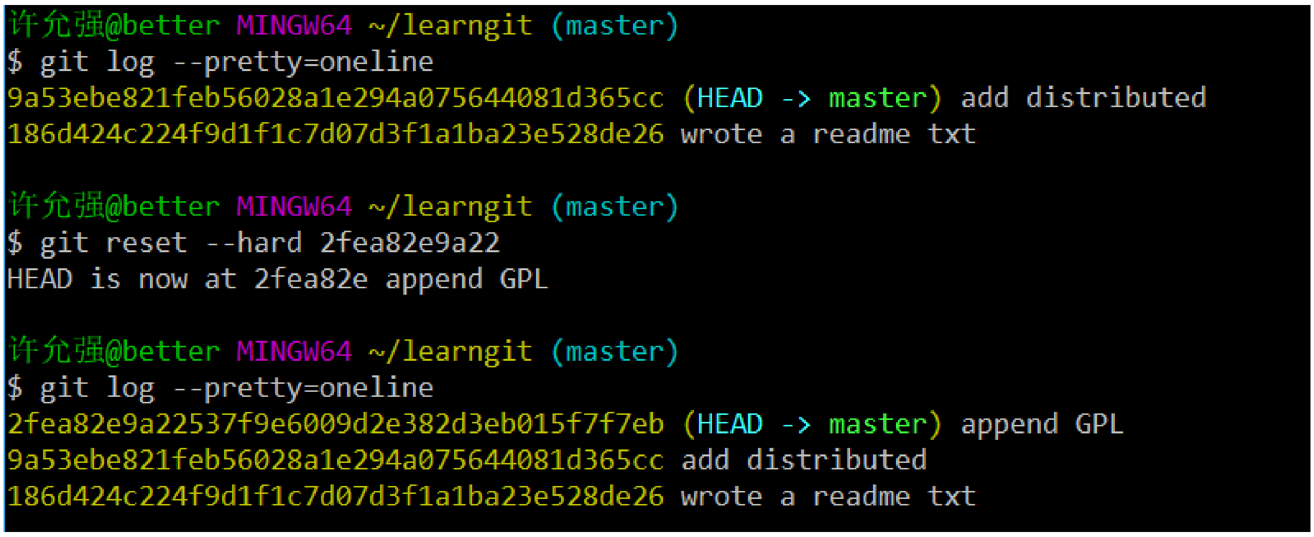
类似`2fea82e9a...`的是`commit id`（版本号）,是一个`SHA1`计算出来的一个非常大的数字，用十六进制表示

6. 在Git中，用`HEAD`表示当前版本，也就是最新的提交`2fea82e9a...`，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`。

使用`git reset`命令可以把当前版本回退



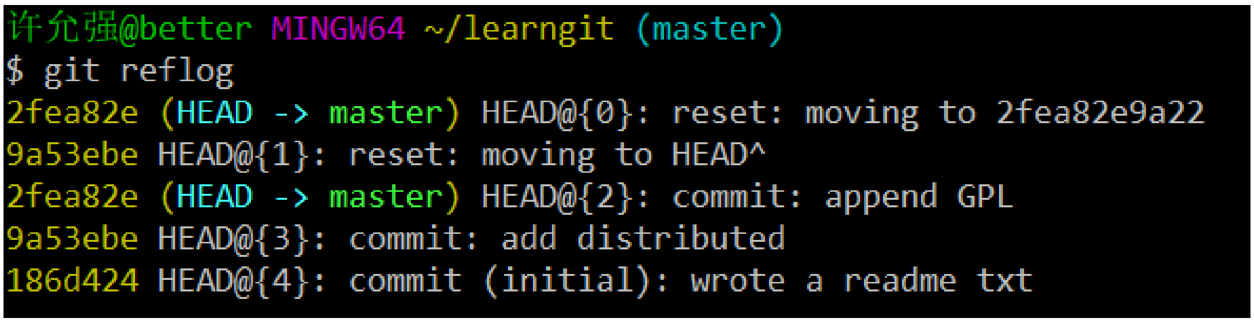
```
许允强@better MINGW64 ~/learngit (master)
$ git log --pretty=oneline
2fea82e9a22537f9e6009d2e382d3eb015f7f7eb (HEAD -> master) append GPL
9a53ebe821feb56028a1e294a075644081d365cc add distributed
186d424c224f9d1f1c7d07d3f1a1ba23e528de26 wrote a readme txt

许允强@better MINGW64 ~/learngit (master)
$ git reset --hard HEAD^
HEAD is now at 9a53ebe add distributed

许允强@better MINGW64 ~/learngit (master)
$ git log --pretty=oneline
9a53ebe821feb56028a1e294a075644081d365cc (HEAD -> master) add distributed
186d424c224f9d1f1c7d07d3f1a1ba23e528de26 wrote a readme txt
```



```
许允强@better MINGW64 ~/learngit (master)
$ git log --pretty=oneline
9a53ebe821feb56028a1e294a075644081d365cc (HEAD -> master) add distributed
186d424c224f9d1f1c7d07d3f1a1ba23e528de26 wrote a readme txt

许允强@better MINGW64 ~/learngit (master)
$ git reset --hard 2fea82e9a22
HEAD is now at 2fea82e append GPL

许允强@better MINGW64 ~/learngit (master)
$ git log --pretty=oneline
2fea82e9a22537f9e6009d2e382d3eb015f7f7eb (HEAD -> master) append GPL
9a53ebe821feb56028a1e294a075644081d365cc add distributed
186d424c224f9d1f1c7d07d3f1a1ba23e528de26 wrote a readme txt
```

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`

7. 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



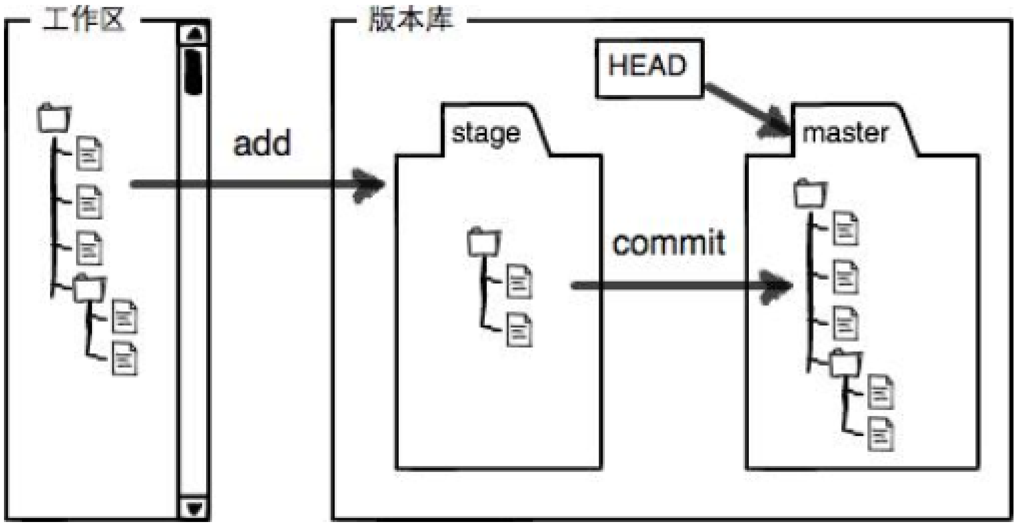
```
许允强@better MINGW64 ~/learngit (master)
$ git reflog
2fea82e (HEAD -> master) HEAD@{0}: reset: moving to 2fea82e9a22
9a53ebe HEAD@{1}: reset: moving to HEAD^
2fea82e (HEAD -> master) HEAD@{2}: commit: append GPL
9a53ebe HEAD@{3}: commit: add distributed
186d424 HEAD@{4}: commit (initial): wrote a readme txt
```

8. **工作区和暂存区**

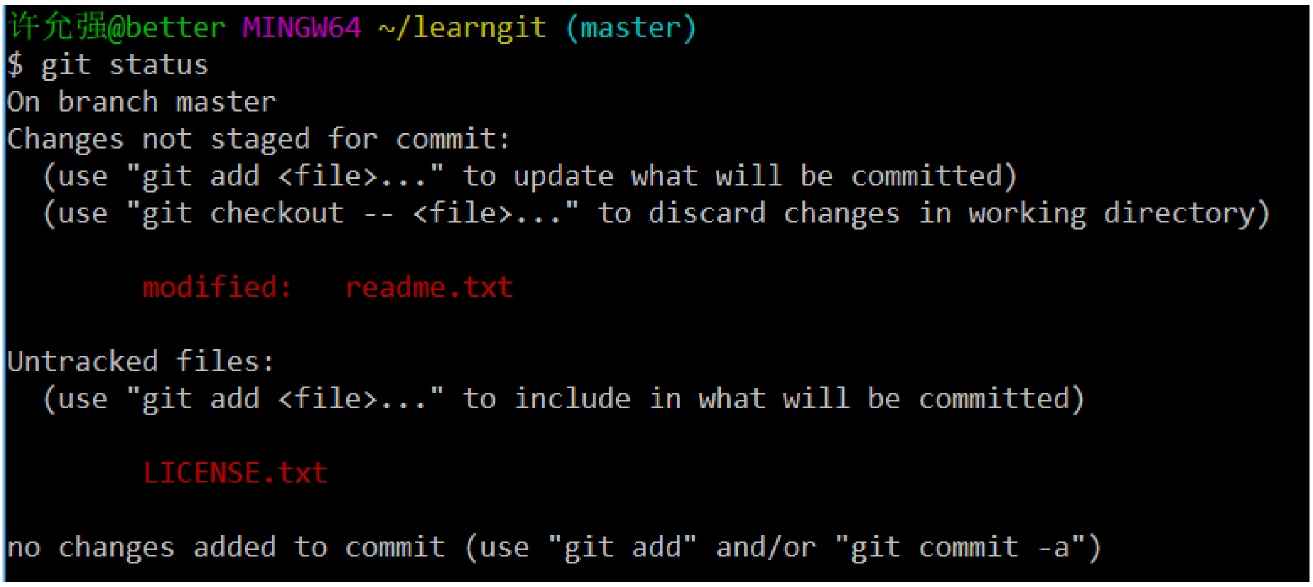
   工作区就是在电脑里能看到的目录，比如`learngit`文件夹就是一个工作区

   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`

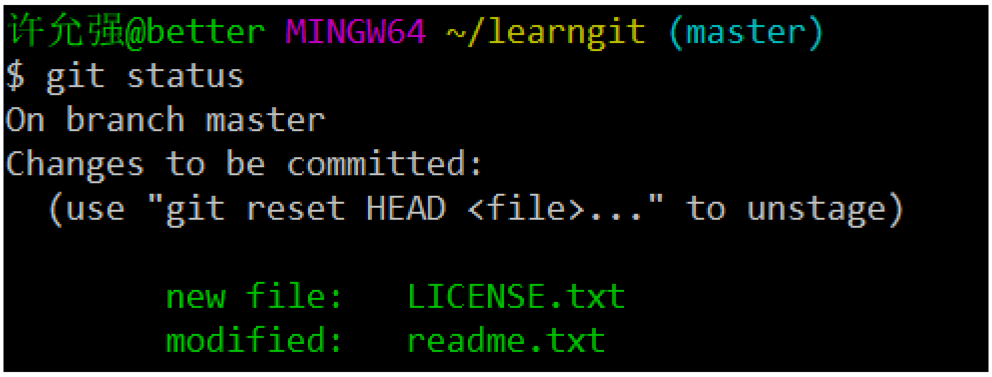


以下 `git status` 命令表示 `readme.txt` 被修改了，而 `LICENSE.txt` 还从来没有被添加过，所以它的状态是 `Untracked`



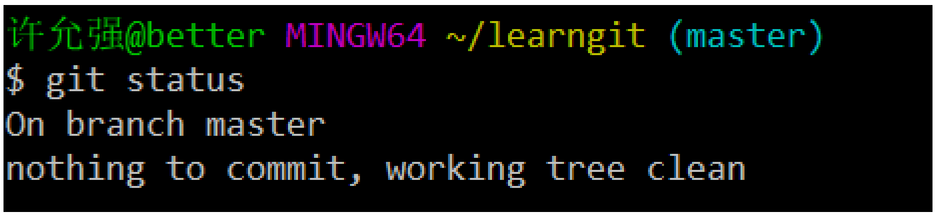
```
许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   readme.txt

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")
```

使用两次命令 `git add`，把 `readme.txt` 和 `LICENSE.txt` 都添加后 `git status` 命令就是把要提交的所有修改放到暂存区（Stage）



```
许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   LICENSE.txt
        modified:   readme.txt
```

执行 `git commit` 就可以一次性把暂存区的所有修改提交到分支,工作区就是“干净”的,暂存区就没有任何内容了



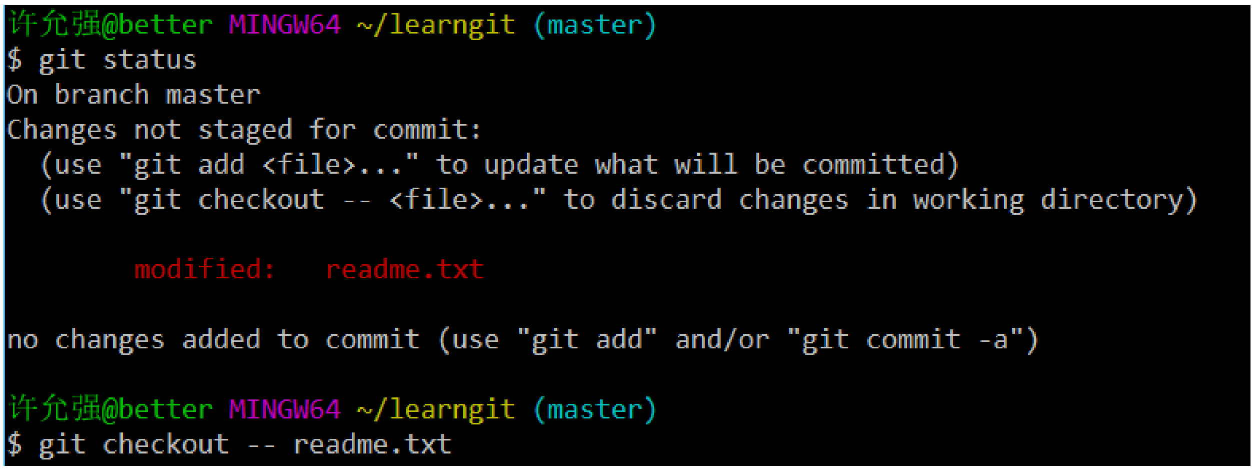
```
许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

9. 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

   git log 很多时需要翻页来看，最后需要按q才能退出

10. **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`。



```
许允强@better MINGW64 ~/learngit (master)
$ 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")

许允强@better MINGW64 ~/learngit (master)
$ git checkout -- readme.txt
```

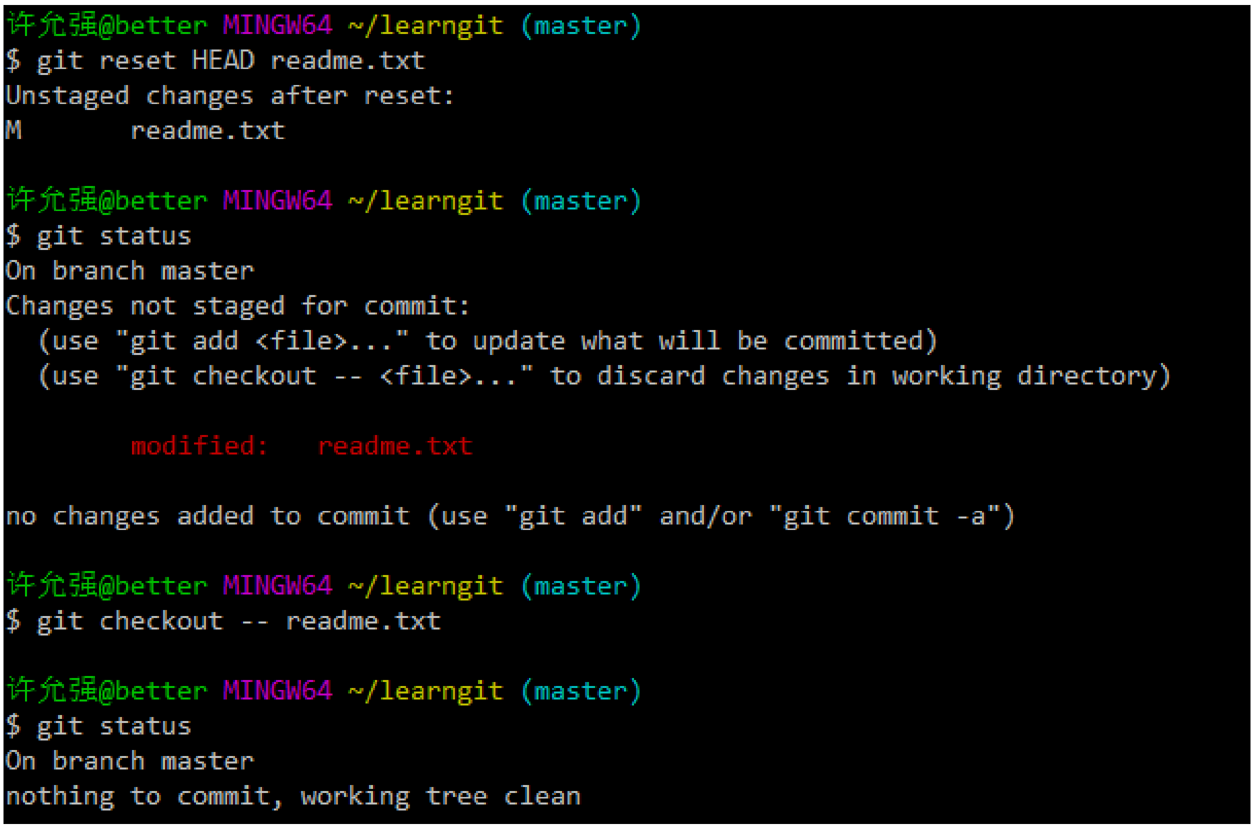
命令`git checkout -- readme.txt`意思就是，把`readme.txt`文件在工作区的修改全部撤销，这里有两种情况：

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。

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。

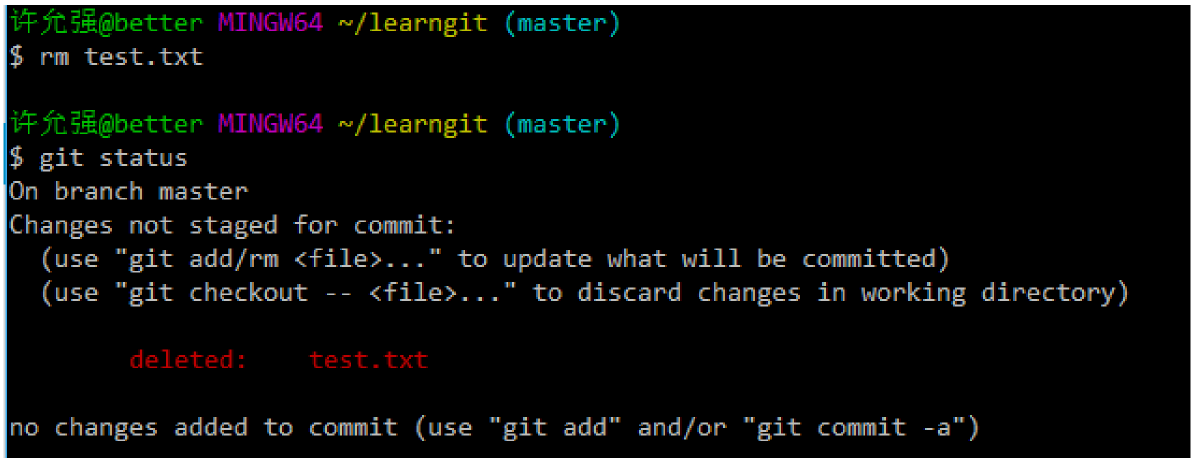


```
许允强@better MINGW64 ~/learngit (master)
$ git reset HEAD readme.txt
Unstaged changes after reset:
M       readme.txt

许允强@better MINGW64 ~/learngit (master)
$ 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")

许允强@better MINGW64 ~/learngit (master)
$ git checkout -- readme.txt

许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

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，使用`git reset --hard`回退，不过前提是没有推送到远程库

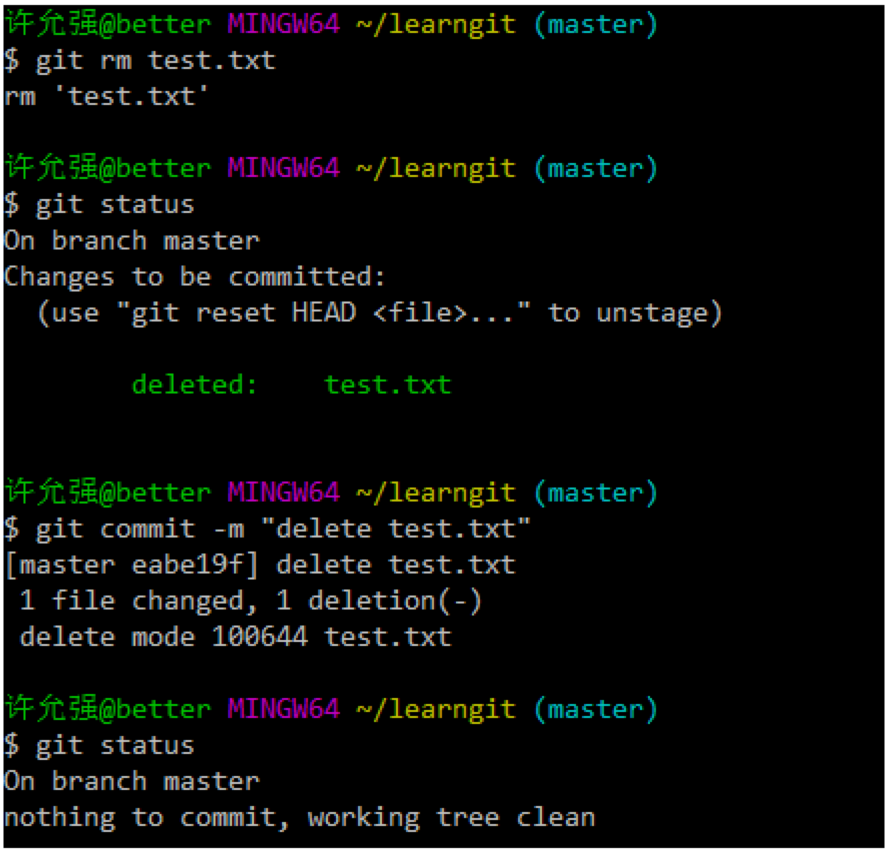
11. **删除文件**

    添加一个新文件`test.txt`到Git并且提交，然后删除`test.txt`，此时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



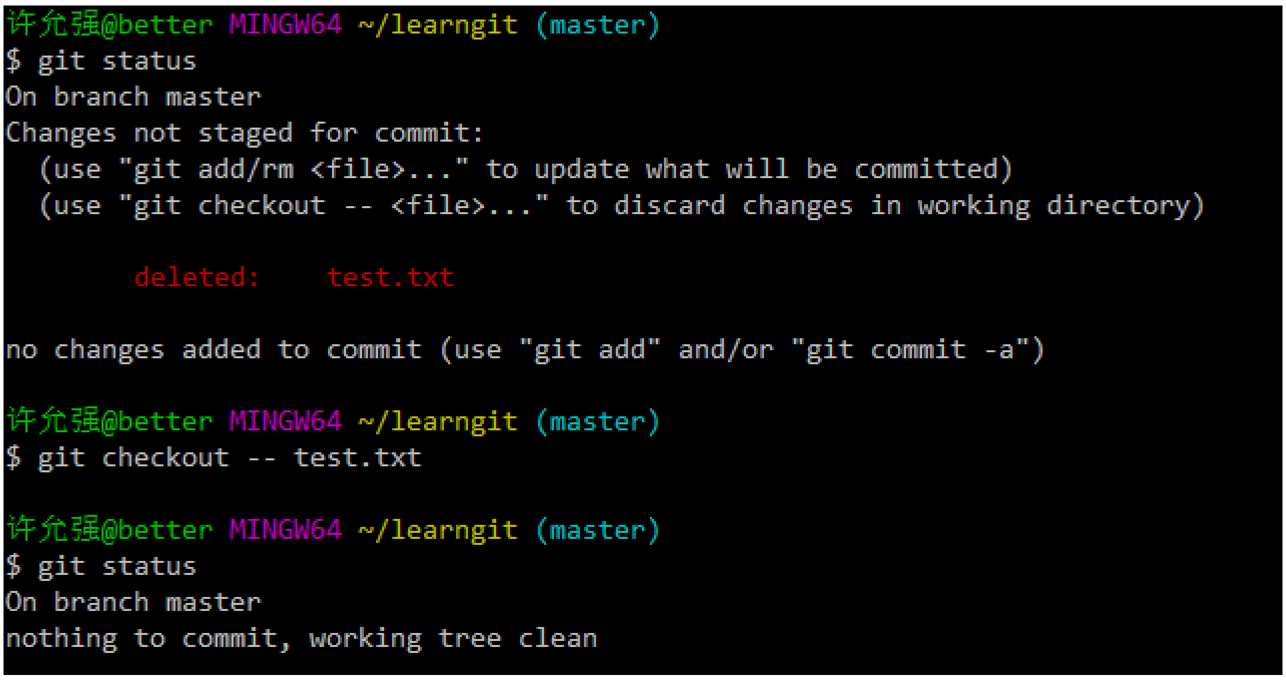
```
许允强@better MINGW64 ~/learngit (master)
$ rm test.txt

许允强@better MINGW64 ~/learngit (master)
$ 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`提交，文件就从版本库中被删除了



```
许允强@better MINGW64 ~/learngit (master)
$ git rm test.txt
rm 'test.txt'

许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    test.txt


许允强@better MINGW64 ~/learngit (master)
$ git commit -m "delete test.txt"
[master eabe19f] delete test.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 test.txt

许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

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



```
许允强@better MINGW64 ~/learngit (master)
$ 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")

许允强@better MINGW64 ~/learngit (master)
$ git checkout -- test.txt

许允强@better MINGW64 ~/learngit (master)
$ git status
On branch master
nothing to commit, working tree clean
```

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。**