git 详解本地操作之一

1. 设置开发者的个人信息

在任何一个系统之中都会存在有多个开发者(多人协作开发)，而在 GIT 之中，对于每一个开发者(电脑)，都需要 开发者自己定义自己的名字与 email 地址，以便进行方便的联系，此时需要配置全局信息。

配置全局用户名及 email 地址

```
git config --global user.name 'wangjinhuai'
git config --global user.email 'wang_jinghuai@163.com'
```

设置完成之后如果成功不会任何提示信息，可以通过如下命令查看全局配置信息：

```
[root@node7 myrpo]# git config -l
user.name=wangjinhuai
user.email=wang_jinghuai@163.com
push.default=simple
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
[root@node7 myrpo]#
```

可以发现除了之前配置的用户名和 email 地址之外，还存在有其它的内容。

2.创建仓库

版本库 = 仓库；
在此仓库中的所有内容都会被 git 管理；
在仓库中的所有文件修改、删除、更新都会被纪录下来；
可以随时恢复到某一特定状态；
初始化仓库： git init

如果要开发项目，那么首先必须有一个仓库(可以简单的理解为是一个磁盘上的文件夹)。

```
[root@node7 ~]# mkdir myrpo
```

此时 mypro 文件夹是一个空的文件夹，没有任何的内容，只是一个纯粹的目录。将 mypro 文件夹定义为仓库，进入文件夹，初始化仓库(将此目录变为可以被 GIT 管理的仓库)

```
[root@node7 ~]# cd myrpo
[root@node7 myrpo]# git init
Initialized existing Git repository in /root/myrpo/.git/
```

而且此时会提示，在 mypro 文件夹之中创建了一个“.git”的目录，这个目录就是仓库信息，

不能修改。

3.添加文件

现在仓库创建完成之后，下面就要进行文件的基本管理了。首先在编写之前有一个说明:所有的文件一定要使用 UTF-8 编码，否则有可能会出现问题。

建立一个 hello.js 文件

```
btn.onclick = function() {
   console.log('每一次新增！');
}
```

查看当前仓库的状态

```
[root@node7 myrpo]# git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Hello.js
nothing added to commit but untracked files present (use "git add" to track)
```

在 "git status" 状态查询操作上可以发展有如下的几个提示信息：
现在开发的属于主分支：On branch master
初始化仓库的提交：No commits yet
未标记的文件：Untracked files:
随后给出的一些操作的命令：(use "git add <file>..." to include in what will be committed)
未标记文件的列表，现在只有一个： hello.js

添加文件到仓库

增加文件到暂存区： git add 文件名称
  提交文件： git commit -m "注释"
  将文件加入到暂存库之中

```
[root@node7 myrpo]# git add hello.js
```

继续查询状态

```
[root@node7 myrpo]# git status
# On branch master
```

```
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   Hello.js
#
```

此时有了一个最重要的信息:

```
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   Hello.js
```

现在的文件并没有真正的提交到主分支上(主分支就是我们真正要运行的程序的所有的代码)。

注意:所有修改的代码都会被 GIT 自动的监测到，所有的代码在使用 commit 提交之前一定要先使用 add 增加进来，否则不会有任何的提交。

如果现在不希望分两步进行则可以在运行以下程序时增加一个“-a”的参数，表示先 add 而后 commit(git commit -a -m "注释")。

提交文件

```
git commit -m "New Js file - Hello.js Create"
```

在进行每次更新提交的时候一般都会为其增加上一些注释数据，所以使用“-m”参数来进行注释的编写。

```
[root@node7 myrpo]# git commit -m "New Js file - Hello.js Create"
[master (root-commit) 89946ed] New Js file - Hello.js Create
 1 file changed, 3 insertions(+)
 create mode 100644 Hello.js
```

此时这个“Hello.js”文件就被真正的提交到了主分支上，也就是意味着程序发布成功了。

查询状态

```
[root@node7 myrpo]# git status
# On branch master
nothing to commit, working directory clean
```

此时的状态会提示:没有任何的信息需要被提交，工作目录很干净。而在 git 工具下用户每一次进行的提交实际上都 会被日志纪录下来。

查看针对于“hello.js”文件的日志信息

```
[root@node7 myrpo]# git log Hello.js
commit 89946ed4d6c53a7846259ce76225e621322e2216
Author: wangjinhuai <wang_jinghuai@163.com>
Date:   Sat Jul 20 04:54:41 2019 -0400

    New Js file - Hello.js Create
```

首先会出现一个提交的信息号 “2e3e7018a965673a4154c84105b5d1a23f13167a”，可以理解为是每一次提交的 id 号。如果有多次提交，那么这个日志信息也会越来越多。

4.修改仓库文件

上面代码已经可以成功的进行了发布，但是代码出现就是为了修改。于是现在来观查对于 git 工具如何去控制修改。

修改 hello.js 文件

```
btn.onclick = function() {
   console.log('每一次新增！');
   console.log('第一次修改·！');
}
```

此时发现文件增加了一行的修改。

查询一下当前的仓库状态

```
[root@node7 myrpo]# 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:   Hello.js
#
no changes added to commit (use "git add" and/or "git commit -a")
[root@node7 myrpo]#
```

现在 GIT 直接提示用户，文件没有保存到暂存区之中，而且提示有:要么你选择文件暂存，要么你直接进行文件的恢复，同时给出了已经修改的文件“hello.js”。

查看文件的前后区别

```
[root@node7 myrpo]# git diff Hello.js
diff --git a/Hello.js b/Hello.js
index 110facc..e7f3e86 100644
--- a/Hello.js
```

```
+++ b/Hello.js
@@ -1,3 +1,4 @@
 btn.onclick = function() {
    console.log('每一次新增！');
+   console.log('第一次修改·！');
 }
```

现在可以发现所有增加的内容都会使用“+”表示，而被删除的信息都会使用“-”表示。
将修改后的代码加入到暂存区后进行提交。

```
[root@node7 myrpo]# git commit -a -m "Update hello.js file. Add one lines"
[master 3cc3d7e] Update hello.js file. Add one lines
 1 file changed, 1 insertion(+)
```

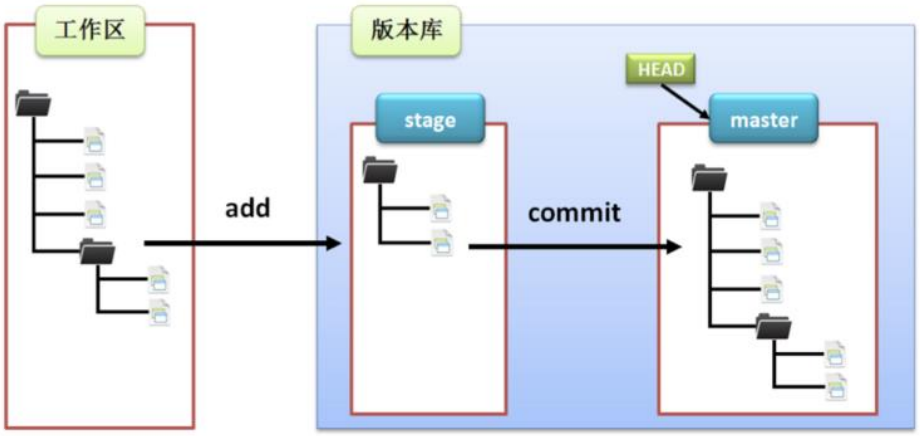
查看修改日志

```
[root@node7 myrpo]# git log Hello.js
commit 3cc3d7e84195333a06acfe2a021fb5ad3746ab8b
Author: wangjinhuai <wang_jinghuai@163.com>
Date:   Sat Jul 20 05:04:24 2019 -0400

    Update hello.js file. Add one lines

commit 89946ed4d6c53a7846259ce76225e621322e2216
Author: wangjinhuai <wang_jinghuai@163.com>
Date:   Sat Jul 20 04:54:41 2019 -0400

    New Js file - Hello.js Create
```

通过以上的代码演示，现在可以清楚的发现，只要是修改的操作 GIT 都可以进行及时的跟踪。

5．工作区与暂存区

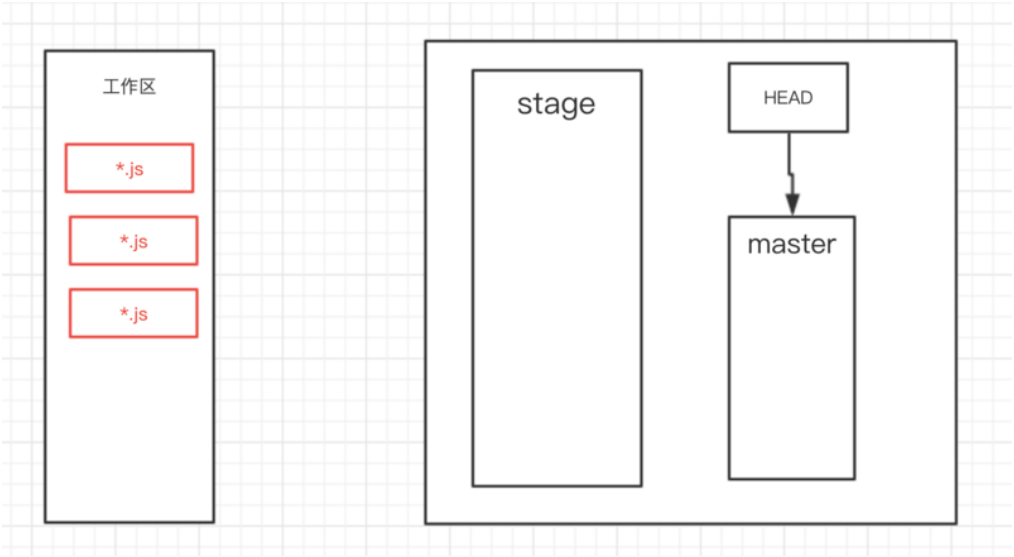
工作区与仓库
工作区： 就是当前电脑的操作目录(包含 .git);
仓库：工作区有一个隐藏目录 .git,这个不算工作区，而是 git 的仓库,git 版本库里保存了很多东西，其中最重要的就是称为 stage 的暂存区，还有 git 为用户自动创建的主程序分支 master ，以及指向 master 的 head 指针。



概念解释：

1) 在之前所编写的“hello.js”文件保存在用户工作区之中；
2) 当使用 add 命令之后，实际上就是将所有的文件提交到暂存区（state） 之中；
3) 使用 commit 命令之后，才表示真正的发出了修改，而真正可以运行的程序都保存在 master 分支上；

6．工作区上的操作



修改 Hello.js

```
btn.onclick = function() {
   console.log('第二次修改·！');
}
```

增加一个 demo.js 文件

```
btn2.onclick = function() {
  console.log('demo click');
```

```
}
```
现在的工作区中的代码已经发生了变化。

用 status 跟踪

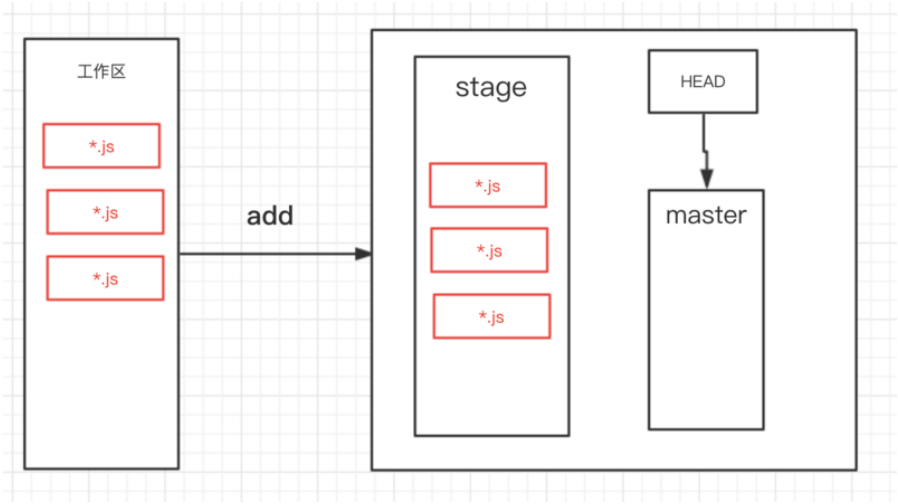
```
[root@node7 myrpo]# git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#     (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   Hello.js
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       demo.js
no changes added to commit (use "git add" and/or "git commit -a")
```

现在会提示有以下信息：

修改了 Hello.js 文件，而这个文件给出了处理方式；

出现了一个未标记的文件(Demo.js)，询问用户是否将其加入到暂存区之中。

7．缓存区上的操作



使用“git add”将代码添加到暂存区之中

```
git add .
```

本次操作使用了一个“.”，那么就表示全部加入。修改之后再次观查状态。

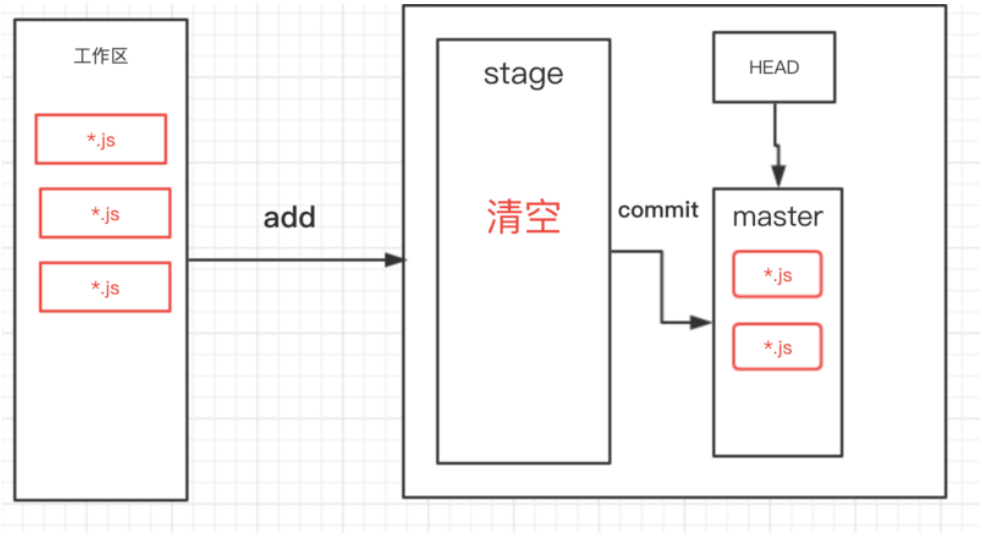
观查状态

```
[root@node7 myrpo]# git status
# On branch master
```

```
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   Hello.js
#       new file:   demo.js
#
```

8．提交修改

数据保存在暂存区之后，下面就要进行代码的提交，将代码提交到主分支上。



当把暂存区的代码提交到主分支上之后，会自动的清空暂存区之中的内容。

提交修改代码

```
[root@node7 myrpo]# git commit -m 'add demo.js file'
[master e765bb5] add demo.js file
 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 demo.js
```

那么此时再次查询状态。

```
[root@node7 myrpo]# git status
# On branch master
nothing to commit, working directory clean
```

那么会直接发现没有任何的文件修改的提示。

9．版本回退

每当用户进行代码提交的时候都会自动的生成一个 commit id，而这个 commit id 就是进行代码回退的主要操作方式。

查询当前修改后的日志信息

```
[root@node7 myrpo]# git log --pretty=oneline
e765bb555cdf9e75613cb4f81d6c2da5781125b1 add demo.js file
3cc3d7e84195333a06acfe2a021fb5ad3746ab8b Update hello.js file. Add one lines
89946ed4d6c53a7846259ce76225e621322e2216 New Js file - Hello.js Create
```

大家可以发现所有的 commit id 并不是顺序的 1、2、3 编号，而是由系统生成一个十六进制数据，这一概念就跟 Session ID 类似，由 GIT 自己控制，主要是为了防止版本号的冲突。

在 master 分之上会有一个 HEAD 指针存在，而这个指针默认情况下永远指向最后一次提交的位置。

当使用回退之后发现 HEAD 指针出现了改变，如果回退一步，那么之前的操作不会被删除，但是所有的代码将回归到指定位置的状态。

**回退一步

```
[root@node7 myrpo]# git reset --hard HEAD~1
HEAD is now at 3cc3d7e Update hello.js file. Add one lines
```

那么如果说现在还想恢复最新的状态呢?那么就必须找到回退的 commit id。

找到所有的已经删除的信息 commitid

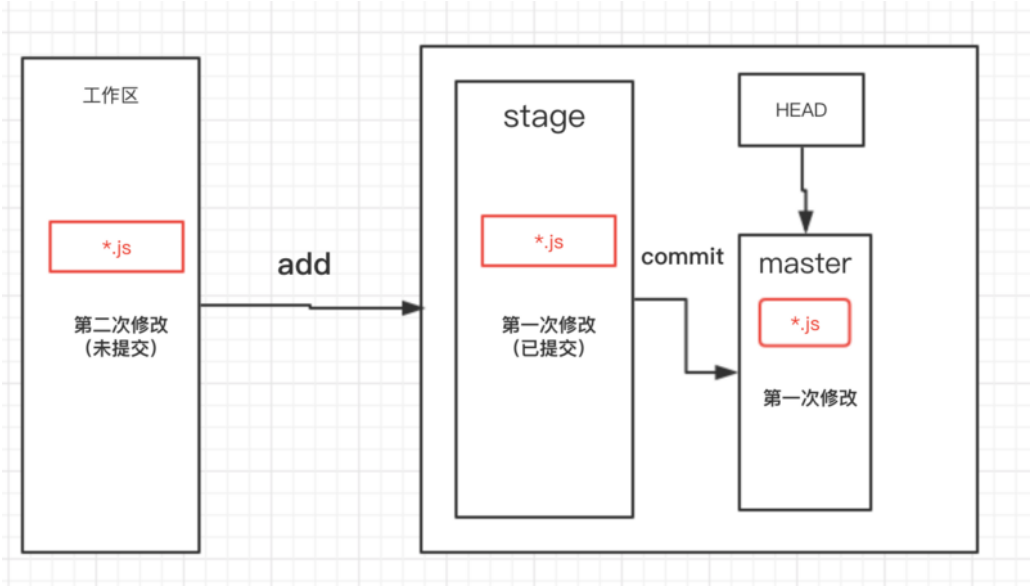
```
[root@node7 myrpo]# git reflog
3cc3d7e HEAD@{0}: reset: moving to HEAD~1
e765bb5 HEAD@{1}: commit: add demo.js file
3cc3d7e HEAD@{2}: commit: Update hello.js file. Add one lines
89946ed HEAD@{3}: commit (initial): New Js file - Hello.js Create
```

恢复最后一次提交

```
[root@node7 myrpo]#  git reset --hard e765bb5
HEAD is now at e765bb5 add demo.js file
```

10.提示:文件修改问题

在有了暂存区和 master 主分支概念之后，就需要回避一个问题:只有保存在暂存区之中的内容才可以被真正的修改， 而不是针对于文件。



编写 hello.js 文件

```
btn.onclick = function() {
   console.log('我的小智');
}
```

以上是 =hello.js 文件的第一次修改。

将修改的文件增加到暂存区之中

```
[root@node7 myrpo]# git add .
```

此时并没有提交，而后再次修改 hello.js 文件。

```
   btn.onclick = function() {
```

```
    console.log('我的王大冶');
  }
```
但是这个时候此文件并没有使用 add 进行加入。

进行提交(提交的时候只提交暂存区的内容)
```
[root@node7 myrpo]# git commit -m "change print"
[master 4d387c6] change print
 1 file changed, 1 insertion(+), 1 deletion(-)
```

可是这个时候只是提交了第一次修改，而第二次修改并没有提交。

查询状态
```
[root@node7 myrpo]# 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:   Hello.js
#
no changes added to commit (use "git add" and/or "git commit -a")
```

进行对比

```
git diff HEAD Hello.js
```

HEAD 是指向最后一次提交的指针，现在的含义是将 HEAD 中的 Hello.java 文件与工作区的 Hello.java 文件进行对比。
```
[root@node7 myrpo]# git diff HEAD Hello.js
diff --git a/Hello.js b/Hello.js
index db11895..2f45d66 100644
--- a/Hello.js
+++ b/Hello.js
@@ -1,4 +1,4 @@
 btn.onclick = function() {
-   console.log('我的小智');
+   console.log('我的王大治');
    console.log('第二次修改·！');
 }
[root@node7 myrpo]#
```
总结:如果一个文件修改多次了，那么就需要执行多次的 add 后才可以提交，否则在 add 前的修改是不会被提交的。

11．撤消修改

情况一：在未增加(git add) 与提交前(git commit) 用户可以直接撤消对文件做出的修改操作。

撤消所做出的修改操作： git checkout -- 文件名
情况二：在已增加（git add）与未提交前(git commit) 用户可以直接撤消对文件所做出的修改操作。

撤消暂存区的修改操作：git reset HEAD 文件名称；
丢掉已经修改的文件内容： git checkout -- 文件名称；

情况一:未增加(git add)&提交(git commit)
如果在工作区之中的代码并没有增加到暂存区之中，那么如果要恢复到原始状态是很容易的。

现在假设修改了 hello.js

```
btn.onclick = function() {
   console.log('我的王大冶');
}
```

就是要改代码，不干走人了，老子不吃你这套
但是只要是文件一修改，那么 git 就可以立即跟踪到状态。

```
[root@node7 myrpo]# 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:   Hello.js
#
no changes added to commit (use "git add" and/or "git commit -a")
```

但是后来发现，此种修改实在是不应该进行，如果进行了，只能有一个结论:此人脑袋有问题。但是写代码的时候 可能不知道上一次修改状态。

恢复

```
git checkout -- hello.js
```

执行之后发现 Hello.java 文件就恢复到了一个原始的状态(上一次的提交状态)。

```
[root@node7 myrpo]# git status
# On branch master
nothing to commit, working directory clean
```

情况二:已增加（ git add ）& 未提交（ git commit ）
现在假设要修改的文件已经提交到了暂存区之中。

将 Hello.java 代码提交到暂存区中

```
[root@node7 myrpo]# git add .
```

当查询状态时:

```
[root@node7 myrpo]#  git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   Hello.js
#
```

在状态查询的时候已经给出了用户的提示，即:你可以根据 HEAD 指针来恢复文件。

从暂存区之中退出

```
git reset HEAD hello.js
```

于是再次查询状态

```
[root@node7 myrpo]#  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)
#
#       modified:   Hello.js
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

相当于现在已经由暂存区中保存的内容恢复到了工作区，那么既然在工作区了，就可以直接恢复原始状态。

恢复原始

```
git checkout -- hello.js
```

个人建议:养成良好的开发习惯，别像演示那样这么对待代码。

12．删除文件

现在在仓库之中存在有 Demo.js 文件，但是假设这个文件从此之后不再使用了呢?只有一个解决方案:删除。但是 在 GIT 里面对于删除文件这一功能严格来讲也属于一个修改操作。

从磁盘上删除 Demo.js 文件

```
[root@node7 myrpo]# rm demo.js
```

当文件删除之后下面查询状态;

```
[root@node7 myrpo]# 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:    demo.js
#
no changes added to commit (use "git add" and/or "git commit -a")
```

这个时候文件是从当前工作区的磁盘中删除了，同时也提示文件被删除。

提交更新

```
git commit -a -m "Delete Demo.java File"
```

但是如果说发现文件被删除错误了呢?则应该进行恢复。

恢复文件

```
git reset --hard bc8e842247b3d78
```

如果文件被删除，则只能够利用版本控制的方式进行恢复。