Git 与 GitHub 入门实践 (/courses/1035)

Git 基本操作

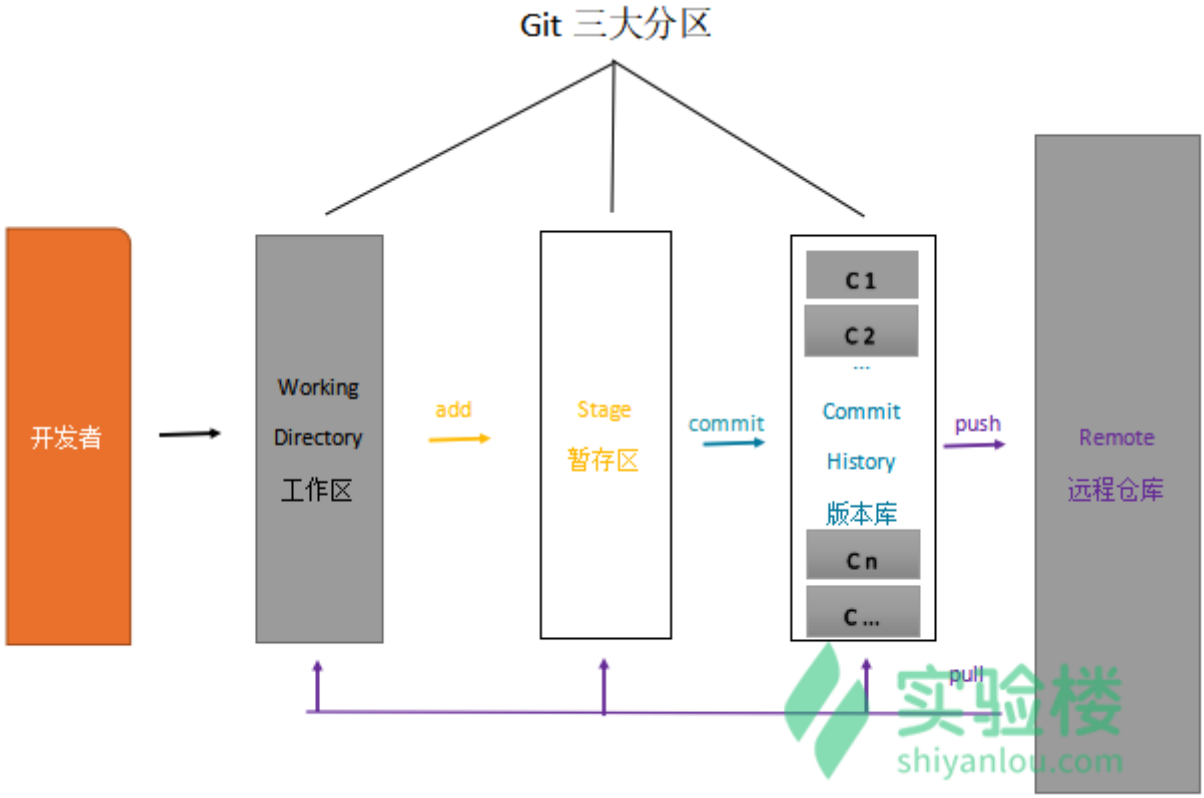
知识点

- Git 仓库的三大区域
- 添加修改到暂存区以及撤销修改
- 查看提交历史
- 配置个人信息
- 版本回退
- 处理 commit 时间线分叉
- 本地仓库 commit 变化记录

一、Git 仓库的三大区域

本节课程我们将完成一次从本地仓库修改代码、提交并推送到远程仓库的操作。

Git 本地仓库有三大区域：工作区、暂存区、版本区。这是一个概念，有这个了解即可，随着使用 Git 的时间增多，慢慢就会理解这三个区域的作用以及为何要这么设计，学习阶段只需按照文档逐步操作即可。接下来我们以命令为主线介绍 Git 的操作。



注意，所有 Git 命令都以 `git` 开头。

二、一次完整的修改、提交、推送操作

首先，进入仓库主目录，执行 `git status` 查看整个仓库的状态：

```
shiyanlou:~/ $ cd shiyanlou
shiyanlou:shiyanlou/ (master) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

无文件要提交，干净的工作区
shiyanlou:shiyanlou/ (master) $
```

实验楼 shiyanlou.com

2.1 对工作区进行修改

创建一个文件并再次查看仓库状态，这步操作是在工作区中：

```
shiyanlou:shiyanlou/ (master) $ ls
README.md
shiyanlou:shiyanlou/ (master) $ echo 'hello world' > one.txt
shiyanlou:shiyanlou/ (master*) $ ls
one.txt  README.md
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

未跟踪的文件:
  （使用 "git add <文件>..." 以包含要提交的内容）

        one.txt

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）
shiyanlou:shiyanlou/ (master*) $
```

实验楼 shiyanlou.com

如上图所示，新建文件后，命令行前缀又发生了一些微小的变化，红色 master 后面出现了 `*` 星号，这表示工作区或暂存区有变化，对文件进行增删改操作都会出现这个星号，另外使用 `git status` 命令亦可查看详情。

2.2 添加修改到暂存区以及撤销修改

按照上图的提示，使用 `git add [文件名]` 命令跟踪此新建文件，即把新增文件添加到暂存区，以备提交：

Git 与 GitHub 入门实践 (/courses/1035)

```
shiyanlou:shiyanlou/ (master*) $ git add one.txt
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

        新文件：   one.txt

shiyanlou:shiyanlou/ (master*) $
```

实验楼 shiyanlou.com

如果对多个文件或目录进行了增删改，可以使用 `git add .` 命令全部添加到暂存区。 未升级 Git 版本的话，可能需要执行 `git add --all` 命令。 注意这里有个概念，当我们修改了工作区， `git add` 命令是将这些修改添加到暂存区，暂存区记录的只是修改。如果要撤销暂存区的修改怎么办？根据上图的提示，执行 `git reset -- [文件名]` 或者 `git rm --cached [文件名]` 命令即可：

```
shiyanlou:shiyanlou/ (master*) $ git reset -- one.txt
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

未跟踪的文件:
  （使用 "git add <文件>..." 以包含要提交的内容）

        one.txt

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）
shiyanlou:shiyanlou/ (master*) $
```

实验楼 shiyanlou.com

上图的命令，如果省略最后的文件名，即可把暂存区的全部修改撤销。好，现在暂存区的修改被撤销，又回到了工作区。

现在介绍另一个命令 `git diff` ，它可以用来查看工作区被跟踪的文件的修改详情，此时新建文件 one.txt 并未被跟踪，而已被跟踪的文件 README.md 无修改，所以看不到。注意，只有在版本区中存在的文件才是被跟踪文件。

我们先修改 README.md 文件，然后执行此命令：

Git 与 GitHub 入门实践 (/courses/1035)

```
shiyanlou:shiyanlou/ (master*) $ ls                                    [13:10:36]
one.txt  README.md
shiyanlou:shiyanlou/ (master*) $ echo 'Git 操作其实很简单，只需多加练习，在实际应用中逐渐领
悟它的设计哲学。' >> README.md
shiyanlou:shiyanlou/ (master*) $ git status                            [13:10:50]
位于分支 master
您的分支与上游分支 'origin/master' 一致。

尚未暂存以备提交的变更：
  （使用 "git add <文件>..." 更新要提交的内容）
  （使用 "git checkout -- <文件>..." 丢弃工作区的改动）

        修改：      README.md

未跟踪的文件:
  （使用 "git add <文件>..." 以包含要提交的内容）

        one.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
shiyanlou:shiyanlou/ (master*) $ git diff                              [13:11:03]
shiyanlou:shiyanlou/ (master*) $                                       [13:11:17]
```

此时会跳到新的页面，即工作区修改详情页，按 `Q` 退出此页面：

```
diff --git a/README.md b/README.md
index ca9bd86..d080bd3 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 # shiyanlou
 Git 课程测试
+Git 操作其实很简单，只需多加练习，在实际应用中逐渐领悟它的设计哲学。
(END)
```

现在，将工作区的两处修改（新增文件 one.txt，修改文件 README.md）全部添加到暂存区，并使用 `git diff --cached` 查看暂存区的全部修改：

```
shiyanlou:shiyanlou/ (master*) $ git add .
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

        修改：      README.md
        新文件：    one.txt

shiyanlou:shiyanlou/ (master*) $ git diff --cached
shiyanlou:shiyanlou/ (master*) $
```

同样，此命令也会跳到新的页面，即暂存区修改详情页：

```
diff --git a/README.md b/README.md
index e69de29..3b18e51 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 # shiyanlou
 Git 课程测试
+Git 操作其实很简单，只需多加练习，在实际应用中逐渐领悟它的设计哲学。
diff --git a/one.txt b/one.txt
new file mode 100644
index 0000000..3b18e51
--- /dev/null
+++ b/one.txt
@@ -0,0 +1 @@
+hello world
(END)
```

实验楼 shiyanlou.com

2.3 查看提交历史

接下来，将执行 `git commit` 命令把暂存区的修改提交到版本区，生成一个新的版本。

在此之前，先介绍另一个命令 `git log` ，它用来查看版本区的提交历史记录，当前只有一个提交，就是在 GitHub 上创建新仓库时的初始化提交。同样此命令也会跳到新页面，如下图所示：

```
commit 2b96af01439c999daee2f6d503e454147e713eb5 (HEAD -> master, origin/master, origin/HEAD
)
Author: Manchangdx <1195581533@qq.com>
Date:   Sun Jan 27 01:26:09 2019 +0800

    Initial commit
(END)
```

实验楼 shiyanlou.com

关于查看提交历史记录的命令，有些常用的选项介绍一下：

- `git log [分支名]` 查看某分支的提交历史，不写分支名查看当前所在分支
- `git log --oneline` 一行显示提交历史
- `git log -n` 其中 n 是数字，查看最近 n 个提交
- `git log --author [贡献者名字]` 查看指定贡献者的提交记录
- `git log --graph` 图示法显示提交历史

2.4 配置个人信息

接下来需要对 Git 进行一些本地配置：

- `user.email` ：写入你自己注册 GitHub 账号的邮箱
- `user.name` ：你自己的 GitHub 账号名字

这两个命令设置你的身份信息。 `git config -l` 可以查看配置信息：

```
shiyanlou:shiyanlou/ (master*) $ git config --global user.email "1195581533@qq.com"
shiyanlou:shiyanlou/ (master*) $ git config --global user.name "Manchangdx"
shiyanlou:shiyanlou/ (master*) $ git config -l
shiyanlou:shiyanlou/ (master*) $
```

[13:55:36]
[13:55:58]
[13:56:15]

Git 与 GitHub 入门实践 (/courses/1035)

完成后，系统自动生成 Git 的配置文件，就是家目录中的隐藏文件 `.gitconfig` ：

```
shiyanlou:shiyanlou/ (master*) $ cat -n ~/.gitconfig
     1  [user]
     2          email = 1195581533@qq.com
     3          name = Manchangdx
shiyanlou:shiyanlou/ (master*) $
```

实验楼 shiyanlou.com

上图所示的配置文件也是可以直接手动修改。

2.5 提交暂存区的修改

现在执行 `git commit` 命令生成一个新的提交，一个必须的选项 `-m` 用来提供该提交的备注：

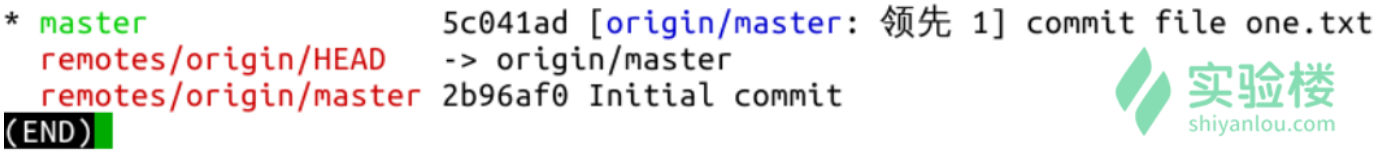
```
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

        修改：     README.md
        新文件：   one.txt

shiyanlou:shiyanlou/ (master*) $ git commit -m 'commit file one.txt'
[master 06f2d6a] commit file one.txt
 2 files changed, 2 insertions(+)
 create mode 100644 one.txt
shiyanlou:shiyanlou/ (master) $ git status
位于分支 master
您的分支领先 'origin/master' 共 1 个提交。
  （使用 "git push" 来发布您的本地提交）

无文件要提交，干净的工作区
shiyanlou:shiyanlou/ (master) $
```

实验楼 shiyanlou.com

提交后，暂存区的修改被清空，执行 `git log` 查看提交记录，紫色框中的十六进制序列号就是提交版本号，这是很重要的信息，每个提交都有自己单独的版本号，就像公民身份证号一样：

观察上图的提交信息，提交版本是按时间倒序排列的，也就是最近的提交排在最上面，你可能需要查看时间正序排列的信息，那么可以使用 `git log --reverse` 命令。

现在介绍一个超级实用、使用频率极高但几乎所有 Git 教程都不重视的命令 `git branch -avv` ，它用来查看全部分支信息。



上图有三行信息，依次说明:

第一行，开头的星号表示当前所在分支，绿色的 master 是分支名，之所以是绿色，也是因为它是当前所在分支。后面第二项是版本号，第三项中括号里面蓝色的字，表示此分支跟踪的远程分支的名字，当然啦，这也是克隆远程仓库到本地时的默认设置 -- 创建 master 分支并自动跟踪远程同名分支；冒号后面黑色文字表示本地分支领先其跟踪的远程分支一个提交。最后一项是提交时填写的备注信息。

第二行，是 Git 指针信息，它指向远程仓库的 master 分支，这行信息暂不重要。

第三行，远程分支信息，详见第一行的解释。

在执行 `commit` 命令时，再介绍一个我并不推荐的选项 `-a` ，它的作用是将未添加到暂存区的修改，也就是工作区的修改也一并提交，但会略过未被跟踪的文件，比如新建文件 one.txt，此命令的完整格式： `git commit -am xxxxx` 。谨慎的做法是按照前文的顺序，修改工作区 - 提交到暂存区 - 随时使用 `git status` 查看仓库状态 - 将暂存区的修改提交到版本区生成一次新的提交。

最后一个环节，将本地新增的提交推送到 GitHub 远程仓库中，命令是 `git push` ，后面不需要任何选项和参数，此命令会把本地仓库 master 分支上的新增提交推送到远程仓库的同名分支上，因为当前所在的分支就是 master，而且上文提到，它已经跟踪了远程仓库的同名分支：

Git 与 GitHub 入门实践 (/courses/1035)

```
shiyanlou:shiyanlou/ (master) $ git push
Username for 'https://github.com': Manchangdx
Password for 'https://Manchangdx@github.com':
枚举对象: 6, 完成.
对象计数中: 100% (6/6), 完成.
使用 4 个线程进行压缩
压缩对象中: 100% (3/3), 完成.
写入对象中: 100% (4/4), 431 bytes | 431.00 KiB/s, 完成.
总共 4 （差异 0），复用 0 （差异 0）
To https://github.com/Manchangdx/shiyanlou.git
   2b96af0..5c041ad  master -> master
shiyanlou:shiyanlou/ (master) $
```

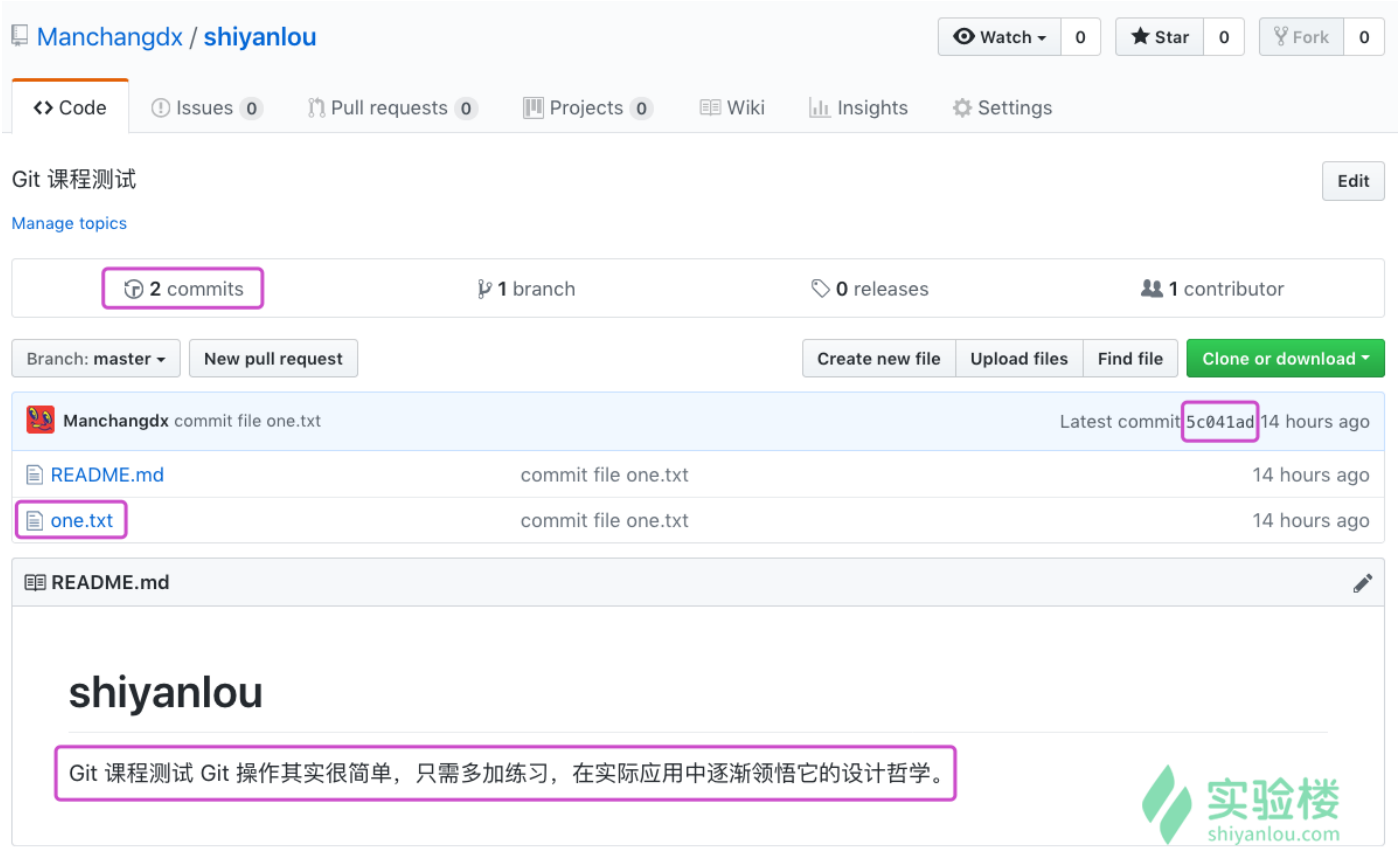
实验楼 shiyanlou.com

此命令需要再次输入用户名和密码，密码为隐藏数据，输入时看不到。推送成功后执行 `git branch -avv` 查看分支情况：

```
* master                5c041ad [origin/master] commit file one.txt
  remotes/origin/HEAD   -> origin/master
  remotes/origin/master 5c041ad commit file one.txt
(END)
```

实验楼 shiyanlou.com

如上图所示，本地分支 master 与远程分支 origin/master 的版本号一致，通常看两个版本号是否一致，只需比对前四位。看一下网页上的情况：



实验楼 shiyanlou.com

完全符合预期。

Git 与 GitHub 入门实践 (/courses/1035)

一个小细节，在上图右侧有 “14 hours ago” 字样，因为这次提交操作是 14 小时前完成的，提交后我睡了一觉，与推送操作的时间无关。

以上就是一次完整的修改 - 提交 - 推送操作。一次推送中可以包含多个 `git commit` 操作，也就是多个提交可以一起推送。

三、版本回退

如果发现 one.txt 文件内容有误，怎么做？可以修改此文件然后再次添加到暂存区、提交、推送，也可以撤销最近一次提交，修改文件后重新提交推送。现在使用后一种方法来演示撤销提交的操作流程。

首先执行 `git reset --soft HEAD^` 撤销最近的一次提交，将修改还原到暂存区。 `--soft` 表示软退回，对应的还有 `--hard` 硬退回，后面会讲到， `HEAD^` 表示撤销一次提交， `HEAD^^` 表示撤销两次提交，撤销 n 次可以简写为 `HEAD~n` 。软退回一个提交后执行 `git branch -avv` 命令查看分支信息：

```
shiyanlou:shiyanlou/ (master) $ git reset --soft HEAD^
shiyanlou:shiyanlou/ (master*) $ git branch -avv
shiyanlou:shiyanlou/ (master*) $
```

```
* master                 2b96af0 [origin/master: 落后 1] Initial commit
  remotes/origin/HEAD    -> origin/master
  remotes/origin/master 5c041ad commit file one.txt
(END)
```

实验楼 shiyanlou.com

可以看到本地仓库的 master 分支的版本号已经发生了变化，变成了前一次提交的版本号，中括号里也有提示信息，本地分支 master 落后其跟踪的远程分支 origin/master 一个提交。

执行 `git status` 查看仓库状态，果然上一个提交中的修改全部扔回了暂存区：

```
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支落后 'origin/master' 共 1 个提交，并且可以快进。
  （使用 "git pull" 来更新您的本地分支）

要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

        修改：      README.md
        新文件：    one.txt

shiyanlou:shiyanlou/ (master*) $
```

实验楼 shiyanlou.com

Git 与 GitHub 入门实践 (/courses/1035)

再次修改 one.txt 文件，执行 `git add .` 命令将新的修改添加到暂存区，然后执行 `git commit` 命令生成新的提交：

```
shiyanlou:shiyanlou/ (master*) $ cat one.txt
hello world
shiyanlou:shiyanlou/ (master*) $ echo 'hello shiyanlou' >> one.txt
shiyanlou:shiyanlou/ (master*) $ cat one.txt
hello world
hello shiyanlou
shiyanlou:shiyanlou/ (master*) $ git add .
shiyanlou:shiyanlou/ (master*) $ git status
位于分支 master
您的分支落后 'origin/master' 共 1 个提交，并且可以快进。
  （使用 "git pull" 来更新您的本地分支）

要提交的变更：
  （使用 "git reset HEAD <文件>..." 以取消暂存）

        修改：     README.md
        新文件：   one.txt

shiyanlou:shiyanlou/ (master*) $ git commit -m 'commit file one.txt'
[master e290005] commit file one.txt
 2 files changed, 3 insertions(+)
 create mode 100644 one.txt
shiyanlou:shiyanlou/ (master) $
```

实验楼 shiyanlou.com

四、处理 commit 时间线分叉

执行 `git status` 和 `git branch -avv` 查看仓库状态和分支状态：

```
shiyanlou:shiyanlou/ (master) $ git status
位于分支 master
您的分支和 'origin/master' 出现了偏离，
并且分别有 1 和 1 处不同的提交。
  （使用 "git pull" 来合并远程分支）

无文件要提交，干净的工作区
```

实验楼 shiyanlou.com

```
* master                 e290005 [origin/master: 领先 1，落后 1] commit file one.txt
  remotes/origin/HEAD    -> origin/master
  remotes/origin/master  5c041ad commit file one.txt
(END)
```

实验楼 shiyanlou.com

可以看到本地仓库的 master 分支与远程仓库的 origin/master 分支在提交版本上有了冲突，又叫做提交时间线分叉。因为刚才的提交操作不是基于远程仓库 origin/master 分支的最新提交版本，而是撤回了一个版本。这种情况下也是可以将本地 master 分支推送到远程仓库的，需要加一个选

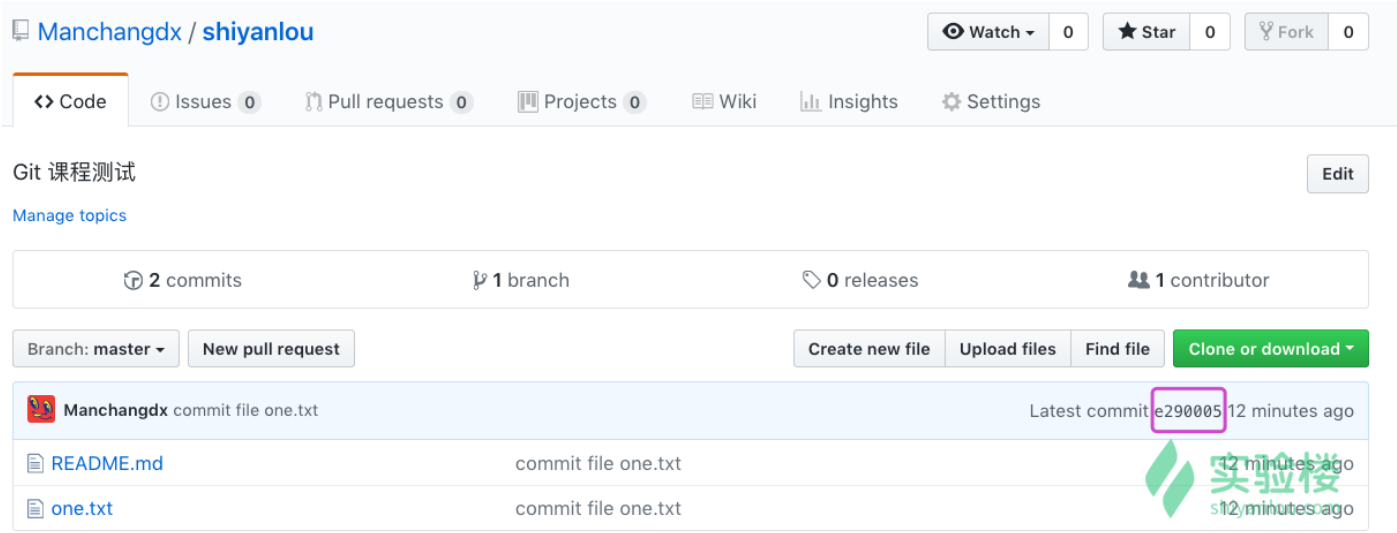
项 `-f` ，它是 `--force` 的简写，这就是强制推送：

Git 与 GitHub 入门实践 (/courses/1035)

```
shiyanlou:shiyanlou/ (master) $ git push -f
Username for 'https://github.com': Manchangdx
Password for 'https://Manchangdx@github.com':
枚举对象: 6, 完成.
对象计数中: 100% (6/6), 完成.
使用 4 个线程进行压缩
压缩对象中: 100% (3/3), 完成.
写入对象中: 100% (4/4), 445 bytes | 445.00 KiB/s, 完成.
总共 4 （差异 0） ，复用 0 （差异 0）
To https://github.com/Manchangdx/shiyanlou.git
 + 5c041ad...e290005 master -> master (forced update)
shiyanlou:shiyanlou/ (master) $ git status
位于分支 master
您的分支与上游分支 'origin/master' 一致。

无文件要提交，干净的工作区
```

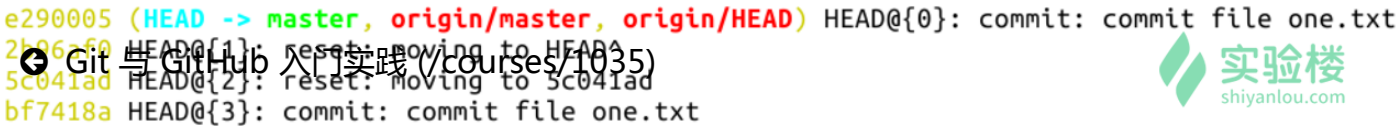
实验楼 shiyanlou.com

执行 `git branch -avv` 看一下分支信息，本地 master 与远程 master 的版本号一致，前四位都是 e290，在浏览器上刷新 GitHub 页面，结果如预期：



五、本地仓库 commit 变化记录

假设此时发现情况不对，之前的那次版本号为 5c04 的提交是正确的，刚才的版本回退操作全都是误操作，怎么办？再次执行一次版本回退吗？当然不需要啦，我们有 `git reflog` 命令，它会记录本地仓库所有分支的每一次版本变化。实际上只要本地仓库不被删除，随你怎么折腾，都能回退到任何地方。 `reflog` 记录只存在于本地仓库中，本地仓库删除后，记录消失。执行此命令如下图所示：



怎么回退到 5c04 那个版本呢？可以直接执行命令 `git reset --hard [版本号]` ，如果记不清版本号，也可以根据上图第 3 行的信息，执行 `git reset --hard HEAD@{2}` 命令，其中 `HEAD@{2}` 就是上图第 3 行第 2 列所示，这个命令的意思是回到当前分支最近两次提交版本变化前：

```
shiyanlou:shiyanlou/ (master) $ git reset --hard HEAD@{2}
HEAD 现在位于 5c041ad commit file one.txt
```

还想反悔，刚才还是改对了，怎么办？再执行一次即可，这次大括号里就是 1 了：

```
shiyanlou:shiyanlou/ (master) $ git reset --hard HEAD@{1}
HEAD 现在位于 e290005 commit file one.txt
shiyanlou:shiyanlou/ (master) $
```

重要的一点，本节全部命令中，只有 `push` 是需要联网执行的，它对远程仓库进行了修改。

六、总结

本节实验包括以下内容的讲解：

- Git 仓库的三大区域
- 修改工作区
- 将工作区的修改添加到暂存区
- 从暂存区撤销修改到工作区
- 查看提交历史
- 配置个人信息
- 完成一次提交
- 版本回退
- 处理提交时间线分叉问题
- 使用 `git reflog` 命令查看本地仓库版本变化

这些内容在实际操作中十分常见，希望大家多多练习，熟练掌握。下节课程，我们将学习设置命令别名、SSH 关联和分支管理的操作。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：Git 与 GitHub 简介 (/courses/1035/labs/7166/document)

下一节：Git 分支操作 (/courses/1035/labs/9816/document)