Git 钩子：定制工作流

一、实验简介

1.1 实验内容

Githook 的简单介绍。hook 的作用域简介。

1.2 实验知识点

- Git

1.3 实验环境

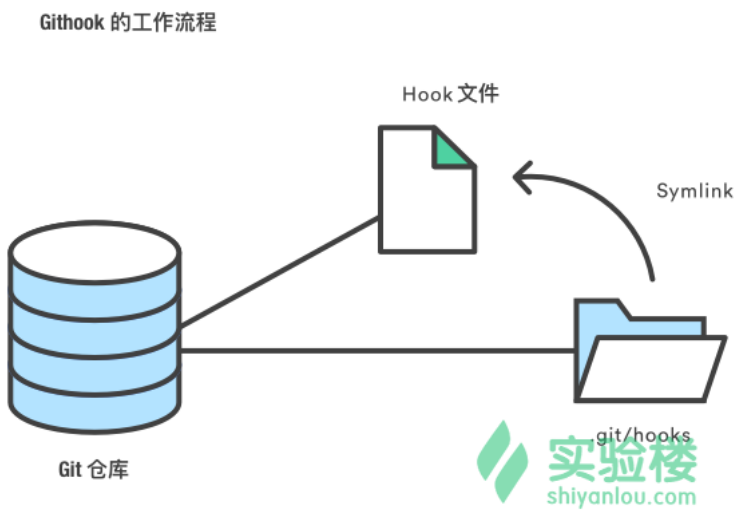
- Xfce终端
- Git CLI

1.4 适合人群

本课程难度属于一般，属于初级级别课程，适合具有 Git 基础的用户。

二、概述

Githook 也称 Git 钩子，是在 Git 仓库中特定事件发生时自动运行的脚本。它可以让你自定义 Git 内部的行为，在开发周期中的关键点出发自定义行为。



Git Hook 最常见的使用场景包括推行提交信息规范，根据仓库状态来改变项目环境，和接入持续集成工作流。但是因为脚本可以完全定制，你可以用 Git Hook 来自动化或者优化你开发工作流中任意部分。

Git Hook 是仓库中特定事件发生时 Git 自动运行的普通脚本。因此 Git Hook 安装和配置也非常容易。Hook 在本地或服务端仓库都可以部署，且只会在仓库中事件发生时被执行。在文章后面我们会具体地研究各种 Hook。

2.1 安装 Hook

Hook 存在与每个 Git 仓库的 `.git/hooks` 目录中。当你初始化仓库时，Git 自动生成这个目录和一些示例脚本。你可以在某个 `.git/hooks` 中，查看这些文件：

```
applypatch-msg.sample      pre-push.sample
commit-msg.sample          pre-rebase.sample
post-update.sample         prepare-commit-msg.sample
pre-applypatch.sample      update.sample
pre-commit.sample
```

这里已经包含了大部分可用的 Hook 了，但是 `.sample` 拓展名防止它们默认被执行。为了安装一个 Hook ，你只需要去掉 `.sample` 拓展名。或者你要写一个新的脚本，你只需添加一个文件名和上述匹配的新文件，去掉 `.sample` 拓展名。

比如尝试安装一个 `prepare-commit-msg` Hook 去掉 `.sample` 扩展名后在文件中写下这两行：

```
#!/bin/sh

echo "# Shiyanlou!" > $1
```

Hook 如果想生效，需要对其修改文件权限，为了确保 `prepare-commit-msg` 可执行，运行下面这个命令：

```
chmod +x prepare-commit-msg
```

接下来每次运行 `git commit` 的时候，你会发现默认的信息被替换了。

2.2 脚本语言

内置的脚本大多是 shell 和 perl 语言实现的，但你也可以使用任何脚本语言，只要它们最后能编译到可执行文件。每次脚本中 `#!/bin/sh` 定义了你的文件将如何被解释。例如我们可以在 `prepare-commit-msg` 中写一个可执行的 Python 脚本。下面这个 Hook 和上一节的 shell 脚本等效：

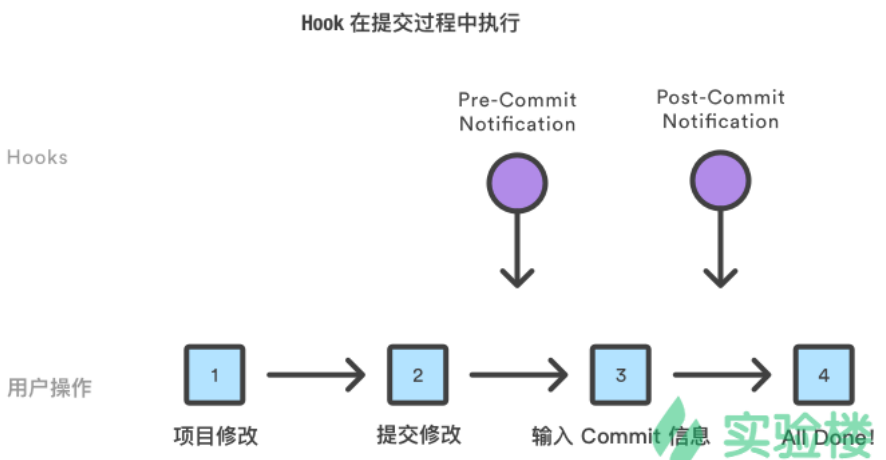
```
#!/usr/bin/env python

import sys, os

commit_msg_filepath = sys.argv[1]
with open(commit_msg_filepath, 'w') as f:
    f.write("# Please include a useful commit message!")
```

> 注意：第一行改成了 Python 解释器的路径。此外 `sys.argv[1]` 用来获取第一个参数。

2.3 钩子的作用域

对于任何 Git 仓库来说 Hook 都是本地的，而且它不会随着 `git clone` 一起复制到新的仓库。而且因为钩子是本地的，任何能接触到仓库的人都可以修改。对于开发团队来说，这有很大的影响。首先，你要确保你们成员之间的钩子都是最新的。其次，你也不能强行让其他人用你喜欢的方式提交——你只能鼓励他们这样做。

在开发团队中维护钩子是比较复杂的，因为 `.git/hooks` 目录不随你的项目一起拷贝，也不受版本控制影响。一个简单的解决办法是把你的 Hook 存在项目的实际目录中（在 `.git` 外）。这样你就可以像其他文件一样进行版本控制。为了安装 Hook ，你可以在 `.git/hooks` 中创建一个符号链接，或者简单地在更新后把它们复制到 `.git/hooks` 目录下。



动手实践是学习 IT 技术最有效的方式！ 开始实验

三、本地 Hook

本地 Hook 只影响它们所在的仓库。以下是最常用的 6 个本地 Hook：

- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- post-checkout
- pre-rebase

前四个 Hook 介入到版本提交的生命周期，后两个允许执行一些额外的操作，分别为 `git checkout` 和 `git rebase` 的安全检查。所有与带 `pre-` 的 Hook 代表即将发生的某个阶段，带 `post-` 只用于通知。

3.1 pre-commit

`pre-commit` 脚本在每次你运行 `git commit` 命令时，Git 向你询问提交信息或者生产提交对象时被执行。你可以用这个 Hook 来价差即将被提交的代码快照。比如说，你可以运行一些自动化测试，保证这个提交不会破坏现有的功能。

3.2 prepare-commit-msg

`prepare-commit-msg` 这个 Hook 在 `pre-commit` Hook 在文本编辑器中生效提交信息之后被调用。 `prepare-commit-msg` 的参数可以是下列三个：

- 包含提交信息的文件名。你可以在原地更改提交信息。
- 提交类型。可以是信息（ `-m` 或 `-F` 选项），模板（ `-t` 选项），merge（如果是个合并提交）或squash（如果这个提交插入了其他提交）。
- 相关提交的SHA1哈希字串。只有当 `-c` ， `-C` ，或 `--amend` 选项出现时才需要。

3.3 post-commit

`post-commit` Hook 在 `commit-msg` Hook 之后立即被运行。它无法改变 `git commit` 的结果，主要用于通知。这里我们详细来讲述一下这个 Hook ，因为我们之后要用到它。

这个脚本没有任何参数，而且退出状态不会影响提交。对于大多数的 `post-commit` 脚本来说，你只是想访问你刚刚创建的提交。你可以用 `git rev-parse HEAD` 来获得最近一次提交的 SHA1 哈希字符串，或者你可以用 `git log -l HEAD` 来获得完整的信息。

3.4 post-checkout

`post-checkout` Hook 和 `post-commit` Hook 很像，但它在你用 `git checkout` 查看引用的时候被调用。

四、实验总结

此章学习了 Git Hook 来修改内部行为的原理，当仓库中特定的时间发生时接收消息。Hook 是存在与 `git/hooks` 仓库中的普通脚本，因此也非常容易安装和定制。

下一节 > (/courses/816/labs/2872/document)