1.1 想用 15 分钟学 Git?

Git 允许用户组同时在同一个文件（甚至代码）里工作，而不干涉他人的工作。它是一个分布式的控制系统。

我们以下的及时终端目前在一个我们决定叫“octobox”的目录里。为了在这里初始化一个 Git 库,打出如下指令：

```
>git init

Initialized empty Git repository in /.git/
```

1.2 检查状态

干的漂亮！就在刚刚 Git 告诉我们，我们的“octobox”目录现在在/.git/.有一个空仓库。这个仓库是 Git 操作的一个隐藏目录文件。

接下来，让我们键入 git status 指令来看我们项目的目前状况。

```
> git status


# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

1.3 添加和提交

我在 octobox 仓库里给你创建了一个叫 octocat.txt 的文件（你可以在下面的浏览器中看到）你需要再次运行 git status 指令来看仓库发生了怎样的改动。

```
$ git status


# On branch master
#
# Initial commit
#
# Untracked files:
#     (use "git add <file>..." to include in what will be committed)
#
#      octocat.txt
nothing added to commit but untracked files present (use "git add" to track)

Success!
```

1.4 添加改动

漂亮，看起来我们的 Git 仓库正在正确的工作。注意到 Git 把 octocat.txt 说成是“untracked”？那意味着 Git 把 octocat.txt 看成一个新文件。

为了告诉 Git 使 octocat.txt 开始追踪改动，我们首先需要用 git add 把它添加到暂存区。

> git add octocat.txt

Nice job, you've added octocat.txt to the Staging Area

1.5 确认改动

干的漂亮！Git 现在正在跟踪我们的 octocat.txt 文件。让我们再次运行 git status 来确认我们的情况。

$ git status

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#     (use "git rm --cached <file>..." to unstage)
#
#      new file:     octocat.txt
#
```

Success!

1.6 提交

注意到 Git 说改动已经被提交？这里列出来的文件已经在暂存区，但是它们还没有在我们的仓库里。我们可以在我们将文件存入仓库之前添加或者移除它们。

为了存储我们的暂存变化，我们运行带有我们已经改动的描述信息的 commit 指令。让我们现在键入并实现它：

$ git commit -m "Add cute octocat story"

```
[master (root-commit) 20b5ccd] Add cute octocat story
 1 file changed, 1 insertion(+)
 create mode 100644 octocat.txt
```

Success!

1.7 添加所有的变化

漂亮！如果你想添加很多同类型的文件，你还可以用通配符。注意我已经把一堆.txt 文件添加

到你下面的目录里面。

我放了一些在“octofamily”的文件里面，另一些选在了我们的“octobox”目录的根目录。幸运的是，我们可以通过用 git add 加通配符来添加我们所有的新文件。别忘了引用！
$ git add '*.txt'

Success!

1.8 提交所有的改动

好，你已经将所有的实验文件添加到了暂存区。随时运行 git status 来看你即将要提交什么。

如果看起来正常，继续运行：
git commit -m 'Add all the octocat txt files'
$ git commit -m 'Add all the octocat txt files'

[master 3852b4d] Add all the octocat txt files
 4 files changed, 4 insertions(+)
 create mode 100644 blue_octocat.txt
 create mode 100644 octofamily/baby_octocat.txt
 create mode 100644 octofamily/momma_octocat.txt
 create mode 100644 red_octocat.txt

Success!

1.9 历史

我们做了一些提交。现在让我们浏览它们来看我们改变了什么。

万幸的是，这里有 git log。你可以将 Git's log 想象成记录我们目前为止所提交的所有变化的日记，并按我们提交的顺序记录。现在试着运行它：
$ git log

commit 3852b4db1634463d0bb4d267edb7b3f9cd02ace1
Author: Try Git <try_git@github.com>
Date: Sat Oct 10 08:30:00 2020 -0500

    Add all the octocat txt files

commit b652edfd888cd3d5e7fcb857d0dabc5a0fcb5e28
Author: Try Git <try_git@github.com>
Date: Sat Oct 10 08:30:00 2020 -0500

Added cute octocat story

Success!

1.10 远端仓库

干的漂亮！我们已经运行并给你创建了一个空 GitHub 仓库来用，在 https://github.com/try-git/try_git.git.试试 Git。为了将我们的本地推送到 GitHub 服务，我们将需要添加一个远端仓库。

这条指令需要用到远程名称和一条仓库的地址，你的仓库地址是 https://github.com/try-git/try_git.git.

继续配合下面的项来运行 git remote add：

$ git remote add origin https://github.com/try-git/try_git.git

Success!

1.11 远端推送

当我们准备好的时候，推送指令会告诉 Git 在哪里放我们的提交，小伙子，我们准备好了。所以让我们把本地改动推送到我们的远端仓库（在 GitHub）

我们的远端叫 origin 并且默认分支名是 master。-u 告诉 tells 去记住参数，所以我们下次可以简单的运行 git push，Git 就能知道该怎么做。继续推送它！
$ git push -u origin master

Branch master set up to track remote branch master from origin.

Success!

1.12 远端更新

让我们假设已经过了一些时间。我们已经邀请其他人来我们的 GitHub，他们更新了你的改动，做了他们自己的提交，并且推送了。
我们可以在我们的 GitHub 仓库里面确认改动并且用一下指令撤销任何任何新改动：
$ git pull origin master

Updating 3852b4d..3e70b0f
Fast-forward

```
 yellow_octocat.txt |      1  +
 1 file changed, 1 insertion(+)
 create mode 100644 yellow_octocat.txt
```

Success!

1.13 差异

看起来我们的 octocat 家族有些添加和修改。让我们用 git diff 指令来看看和我们上次提交的有什么差异。

在这种情况下我们想看和最近的提交的差异，我们可以参考使用 HEAD 指针。

```
$ git diff HEAD
```

```
diff --git a/octocat.txt b/octocat.txt
index 7d8d808..e725ef6 100644
--- a/octocat.txt
+++ b/octocat.txt
 @@ -1 +1 @@
 -A Tale of Two Octocats
 + [m A Tale of Two Octocats and an Octodog
```

Success!

1.14 暂存差异

diff 的另外一种好的用法是我已查看我们已经暂存了的改动。记住，暂存文件是我们已经告诉 git 已经准备提交的文件。

让我们使用 git add 来暂存 octofamily/octodog.txt,我刚刚帮你把它添加到了家族里面。

```
$ git add octofamily/octodog.txt
```

Success!

1.15 暂存差异（cont'd)

漂亮 ,现在继续运行 git diff 加--staged 参数来看你刚刚暂存的改动。你应该看到 octodog.txt 已经被创建了。

```
$ git diff --staged
```

```
diff --git a/octofamily/octodog.txt b/octofamily/octodog.txt
new file mode 100644
```

index 0000000..cfbc74a
--- /dev/null
+++ b/octofamily/octodog.txt
@@ -0,0 +1 @@
+ [m woof

Success!

1.16 重置暂存

现在 octodog 已经是家族的一部分 ,octocat 都非常沮丧。因为我们爱 octocat 胜过 octodog，我们将移除 octodog.txt 来安慰它。

你可以用 git reset 指令删除暂存文件。继续，移除 octofamily/octodog.txt。

$ git reset octofamily/octodog.txt

Success!

1.17 撤销

git reset 已经好好的撤销了暂存 octodog.txt，但是你会注意到它任然在哪里。他只是不再暂存了。如果我们可以回到 octodog 过来毁掉宴会之前，那真是太棒了。

文件可以用指令 git checkout -- <target> 让它们回到最新的提交。继续并且甩掉到 octocat.txt 的最新一次提交的所有改动。

$ git checkout -- octocat.txt

Success!

1.18 分支

当开发者在一个属性或漏洞工作时，他们经常给他们的代码创建一个副本（aka. branch）来使提交独立。当他们完成之后，它们可以合并这些分支到他们的主分支。

我们想移除所有这些讨厌的 octocat,所以让我们创建一个叫做 clean_up 的分支,所有的工作将在这里进行:

$ git branch clean_up

Success!

1.19 切换分支

漂亮！现在如果你键入 git branch 你将看到两条本地分支：一条叫 master 的主分支和你的叫 clean_up 的新分支。

你可以用 git checkout <branch>指令来切换分支。现在试着切换到 clean_up 分支：

$ git checkout clean_up

Switched to branch 'clean_up'

Success!

1.20 移除所有的东西

好，现在你在 clean_up 分支。你终于可以用 git rm 指令移除所有讨厌的 octocat，它不仅从磁盘删除实际的文件，并且会给我们暂存删除的文件。

你将再次用通配符在一次扫描中获得所有的 octocats,继续运行：

$ git rm '*.txt'

rm 'blue_octocat.txt'
rm 'octocat.txt'
rm 'octofamily/baby_octocat.txt'
rm 'octofamily/momma_octocat.txt'
rm 'red_octocat.txt'

Success!

1.21 提交分支改动

现在你既然已经移除了所有的 cat,你将需要提交的你的改动。

随时运行 git status 来查看你已经提交的改动。

$ git commit -m "Remove all the cats"

[clean_up 63540fe] Remove all the cats
 5 files changed, 5 deletions(-)
 delete mode 100644 blue_octocat.txt
 delete mode 100644 octocat.txt
 delete mode 100644 octofamily/baby_octocat.txt
 delete mode 100644 octofamily/momma_octocat.txt
 delete mode 100644 red_octocat.txt

Success!

1.22 切换回 master

漂亮！你几乎处理完了所有的猫……呃,漏洞修复,你只需要切换回 master 分支，那样你可以把你的改动从 clean_up 分支复制（或者合并）到 master 分支。

继续并确定 master 分支：

```
$ git checkout master
```

```
Switched to branch 'master'
```

Success!

1.23 准备合并

好，已经到了你要把 clean_up 分支合并到 master 分支的时刻了。深呼吸,没有那么恐怖。

master 分支已经准备好了,所以我们只需要告诉 Git 把 clean_up 分支合并到它里面：

```
$ git merge clean_up
```

```
Updating 3852b4d..ec6888b
Fast-forward
 blue_octocat.txt                 |     1 -
 octocat.txt                      |     1 -
 octofamily/baby_octocat.txt      |     1 -
 octofamily/momma_octocat.txt     |     1 -
 red_octocat.txt                  |     1 -
 5 files changed, 5 deletions(-)
 delete mode 100644 blue_octocat.txt
 delete mode 100644 octocat.txt
 delete mode 100644 octofamily/baby_octocat.txt
 delete mode 100644 octofamily/momma_octocat.txt
 delete mode 100644 red_octocat.txt
```

Success!

1.24 保持干净

恭喜！你刚刚成功的完成了你的第一次漏洞修复和合并。你接下来只需要清洁你自己。因为你已经处理了 clean_up 分支，并且你已经不再需要它了。

你可以用 git branch -d <branch name>指令来删除一条分支。继续并删除 clean_up 分支：

```
$ git branch -d clean_up
```

```
Deleted branch clean_up (was ec6888b).
```

Success!

1.25 最后的推送

现在 ,我们到了最后一步。你将它进行到了这一步,并且它能成为你学习 Git 的好的一课,我感到非常的自豪 。你接下来所需要做的只是将你工作的所有工作推送到你的远端仓库,做吧！

```
$ git push


To https://github.com/try-git/try_git.git
    3e70b0f..8c3985d    master -> master

Success!
```