tryGit练习

# 1.1用15分钟如何学习Git

Git能够让团队在同一时间同一文档下进行工作，不会踩到别人的脚趾，互不干扰。Git是一个分布式的版本控制系统我们下面的终端命令当前所在目录为我们称之“octobox”的目录。为了在这初始化一个Git仓库，输入以下命令：

git init

# 1.2检查状态

很好！Git刚提示我们，我们的“octobox”目录下现在有了一个空仓库在/.git/。这个仓库是一个隐藏的目录，Git在其中进行操作。当你通过这个指导，想要保存你的进度和想当你成功完成后想获得一枚徽章，那就去创建一个免费的Code School账号吧。我们会在这里等你。接下来，让我们输入git sataus 命令来看看我们的项目现在的状态是什么：

git status

# 1.3添加&提交

我在“octobox”仓库为你创建了一个命名为“octocat.txt”的文件。（你可以在浏览器下面看到）

你应该再次运行一次git status命令，看看仓库的状态有什么改变。

# 1.4添加改变

好滴，这看起来我们的Git仓库正在恰当地工作着。注意注意Git是如何说octocat.txt是“不露痕迹”？这意味着Git知道octocat.txt是一个新的文件。

# 1.5 检查改变

很好! Git现在可以跟踪octocat.txt文件啦。让我们再次运行一次git status 看看我们现在所处的地方：

git status

# 1.6 提交

注意Git是怎么描述改变被提交的？ 这里所列的这些文件现在在一个暂存区，然而它们还没有存在于我们的仓库。在我们将它们存放在仓库之前，我们可以从暂存区增加或者移除文件。为了存放我们的阶段性变化，我们可以运行commit命令并附上信息描述我们做了什么修改。现在让我们试一下，输入一下命令：

git commit -m "Add cute octocat story"

# 1.7 添加全部改变

很好！如果你想添加许多相同类型的文件，你也可以使用通配符。注意到我已经添加了一些.txt后缀类型的文件在你的目录下面。

我放了一些在名为“octofamily”的目录下和一些在我们根目录“octobox”下。幸运的是，我们可以使用git add加上通配符添加所有新的文件。不要忘了单引号！

git add '\*.txt'

# 1.8 提交所有改变

好的，你已经将所有的文本文件添加到了暂存区。键入

git status命令查看你准备要提交什么。如果看上去没什么问题，继续并运行:

git commit -m 'Add all the octocat txt files'

# 1.9 历史

所以我们已经做了一些提交了。现在让我们浏览一下我们做的改变。

我们很幸运，有一条git log命令。认为Git的日志就好像日记一样记得目前为止我们提交的所有改变，在这一顺序下我们提交它们。现在试着运行一下：

git log

# 1.10 远端仓库

干的不错！我们已经提前为你创建了一个新的空GitHub仓库，让你能使用在https://github.com/try-git/try\_git.git站点上的Try Git。 要推送本地改动到GitHub服务器，我们需要添加一个远端仓库。

这个命令需要一个远程名和远程服务器仓库URL，在这个练习里是https://github.com/try-git/try\_git.git。继续并运行 git remote add 加上以下的选项：

git remote add origin https://github.com/try-git/try\_git.git

# 1.11 推送远程服务器

这个推送命令告诉Git当我们准备好后那里推送我们的提交。所以让我们把我们的本地改动推送到远程服务器上面。（GitHub上）

我们的远程服务器的名字是origin和默认本地分支的名字是master。- u告诉Git记住参数,以便下次我们可以简单地运行git push和Git将知道该怎么做。继续推送它!

git push -u origin master

# 1.12 Pulling Remotely

假设一段时间已经过去了。我们邀请别人到GitHub，他们下载您的更改，,做出了自己的提交,并推送它们。我们可以在GitHub库检查更改和下载任何新的变化，通过运行以下命令:

git pull origin master

# 1.13差异

哦，看起来有一些补充和修改的octocat家庭。让我们看看与我们的最后一次用git diff命令提交有什么不同。

在这种情况下，我们希望查看我们最近提交的差异，我们可以使用头指针。

git diff HEAD

# 1.14阶段的差异

diff的另一个伟大的使用就是可以观察已经被我们暂存的文件的改变。记住，我们已经告诉Git暂存的文件是准备提交的文件。

让我们用git add阶段octofamily / octodog.txt，我刚刚加入你的家庭。

git add octofamily / octodog.txt

1.15阶段的差异（续）

好的，现在运行git diff --staged,看看你刚暂存的有什么改变。你应该看到，octodog.txt创建。

git diff --staged

# 1.16重置阶段

所以现在octodog是家庭的一部分，octocat都郁闷。因为我们爱octocat超过octodog，我们会让他舒展眉毛通过去除octodog.txt。

你可以不暂存文件通过使用git reset命令。请删除octofamily / octodog.txt。

git reset octofamily / octodog.txt

git checkout -- octocat.txt

# 1.17撤销

git reset做了伟大的工作，去除octodog.txt的暂存，但你会发现他还在那里。他只是不在暂存区了。这将是伟大的如果我们能回到在octodog来到了聚会前的过去。

文件可以变回他们最后提交的命令：git checkout——<目标>。去摆脱所有的变化自上次提交octocat.txt

git checkout -- octocat.txt

# 1.18分支

当开发人员正在开发一个功能或调试错误时，他们通常会创建一个副本（也就是分支）的代码，他们可以单独提交。然后，当他们做的，他们可以把这一分支合并主分支。

我们要消除这些恼人的octocats，让我们创建一个分支称为clean\_up，在那里我们会做所有的工作：

clean\_up分支

# 1.19切换分支

太好了!如果你现在输入gitbranch你会看见两个分支：主要分支命名为master和你的新分支命名为clean\_up。

你可以用git checkout <branch>命令。现在尝试切换到clean\_up分支：

git checkout clean\_up

# 1.20删除所有的东西

好的，所以你在clean\_up分支。你终于可以用git rm命令，不仅将从磁盘删除实际文件删除所有那些讨厌的octocats，但也将我们的暂存区的文件也删除。

你想再次使用通配符的一次扫描中得到所有的octocats，继续运行：

git rm \* .txt”

# 1.21提交分区改变

现在，你已经删除了所有的猫，你需要提交的变化。

随时运行git status检查你正要提交的更改。

git commit -m“Remove all the cats”

git checkout master

# 1.22切换回主

好，你几乎完成关于猫的bug修复，你只需要切换到主分支，你就可以从clean\_up分支到主分支复制（或合并）你的改变。

去，切换到主分支：

git checkout master

# 1.23准备合并

好了，时间到了的时候你要从clean\_up分支到主分支合并更改。深呼吸，这不是可怕的。

我们已经在主分支，所以我们只需要告诉git将clean\_up分支合并到它那：

# git merge clean\_up

# 1.24保持清洁

祝贺你!你刚刚完成了你的第一个成功的修正和合并。所有剩下的都是你自己清理干净。因为你做的clean\_up分支你不需要它了。

你可以用git branch -D <公司名称>删除分支。继续并删clean\_up分支：

git branch -d clean\_up

# 1.25最后推送

在这里，我们在最后一步。我很自豪，你已经做的很不错。你现在所做的一切都是为了推送你一直致力于你的远程仓库的所有工作，并且你都完成了！

git push