1.1 Got 15 minutes and want to learn Git?

Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

Git 允许一群人同时在同一个文档（通常是代码）工作而又不会互相干扰。是一个分布式版本的控制系统。

Our terminal prompt below is currently in a directory we decided to name "octobox". To initialize a Git repository here, type the following command: git init

下面终端的提示是我们现在决定命名为“octobox”的目录。输入下面的命令“get init”来初始化一个Git仓库。

# 1.2 Checking the Status

Good job! As Git just told us, our "octobox" directory now has an empty repository in /.git/. The repository is a hidden directory where Git operates.

正如Git告诉我们的，目录“octobox”现在有一个空仓库在/.git/。Git操作的这个空仓库是一个隐藏目录。

To save your progress as you go through this tutorial -- and earn a badge when you successfully complete it -- head over to[create a free Code School account](https://www.codeschool.com/account/courses/try-git/add). We'll wait for you here.

当你经过这个目录的时候保存你的进程——当你成功地建立一个自由的代码项目的时候你会赢得一个勋章。

Next up, let's type the git status command to see what the current state of our project is: git status

接下来，让我们输入命令“git status”来查看我们的项目现处于哪个状态。

# 1.3 Adding & Committing

I created a file called octocat.txt in the octobox repository for you (as you can see in the browser below).

我创造了一个名为“octocat.txt”的文件在“octobox”仓库给你（正如你在下面浏览器所看到的。）

You should run the git status command again to see how the repository status has changed:

你可以重新执行命令“git status”来看看仓库状态是怎么变化的。

# 1.4 Adding Changes

Good, it looks like our Git repository is working properly. Notice how Git says octocat.txt is "untracked"? That means Git sees that octocat.txt is a new file.

看起来我们的Git仓库在正常的工作。注意为什么Git显示“octocat.txt”是无路径的？这是因为Git 把“octocat.txt”看成是一个新文件。

To tell Git to start tracking changes made to octocat.txt, we first need to add it to the staging area by using git add. Git add octocat.txt

为了让Git创造一个路径到ocotcat.txt，我们需要第一次用命令“git add”把它添加到暂存区域。

# 1.5 Checking for Changes

Good job! Git is now tracking our octocat.txt file. Let's run git status again to see where we stand: git status

Git 现在建立了到文件“octocat.txt”的路径，让我们再次执行“git status”来看看我们处于哪个状态。

# 1.6 Committing

Notice how Git says changes to be committed? The files listed here are in the Staging Area, and they are not in our repository yet. We could add or remove files from the stage before we store them in the repository.

注意此时Git与尚未提交之前的变化。这里列出的文件已经不在我们建立的仓库里面了，而是存在于暂存区域。我们可以添加或者移动暂存区域中的文件在我们将它们存储于仓库之前。

To store our staged changes we run the commit command with a message describing what we've changed. Let's do that now by typing:

Git commit -m “Add cute octocat story”

为了保存我们暂存区域中的变化，我们执行一个描述我们变化的消息的命令（提交文件），我们可以通过输入Git commit -m “Add cute octocat story”来完成。

# 1.7 Adding All Changes

Great! You also can use wildcards if you want to add many files of the same type. Notice that I've added a bunch of .txt files into your directory below.

你也可以使用通配符如果你想添加很多相同格式的文件。注意我已经添加了一堆.txt格式的文件到你的目录中（在下面指令行有显示）。

I put some in a directory named "octofamily" and some others ended up in the root of our "octobox" directory. Luckily, we can add all the new files using a wildcard with git add. Don't forget the quotes!

Git add ‘\*.txt’

我放了一些文件在“octofamily”这个目录下，还有其他的文件放在你的根目录“octobox”下。幸运的是，我们可以使用通配符来添加所有的新文件，通过指令“git add”来完成。别忘了加上引号。

# 1.8 Committing All Changes

Okay, you've added all the text files to the staging area. Feel free to run git status to see what you're about to commit.

If it looks good, go ahead and run:

git commit -m ‘add all the octocat txt files’

你已经添加了所有的text文件到暂存区域。尝试运行“git status”来看看你将要提交什么。如果结果看起来正确，接下去运行 git commit -m ‘add all the octocat txt files’

# 1.9 History

So we've made a few commits. Now let's browse them to see what we changed.

Fortunately for us, there's git log. Think of Git's log as a journal that remembers all the changes we've committed so far, in the order we committed them. Try running it now: git log

目前我们已经进行了一些提交。现在让我们来浏览一下我们改变了什么。幸运的是，我们有指令“git log”。“git log”就像日记一样记录了我们目前为止提交产生的所有改动。试试运行“git log”。（对commit的翻译不够准确。）

# 1.10 Remote Repositories

Great job! We've gone ahead and created a new empty GitHub repository for you to use with Try Git at https://github.com/try-git/try\_git.git. To push our local repo to the GitHub server we'll need to add a remote repository.

This command takes a remote name and a repository URL, which in your case is https://github.com/try-git/try\_git.git.

Go ahead and run git remote add with the options below:

Git remote add origin <https://github.com/try-git/try_git.git>

我们继续为你使用 “try git”在这个网站上建立一个新的空仓库。为了推动我们当地的repo到Github服务器，我们需要添加一个远程仓库。（repo，仓库定义文件名的结束用）

这个命令产生一个远程仓库的名字跟URL。

# 1.11 Pushing Remotely

The push command tells Git where to put our commits when we're ready, and boy we're ready. So let's push our local changes to our origin repo (on GitHub).

The name of our remote is origin and the default local branch name is master. The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do. Go ahead and push it!

git push -u origin master

push命令告诉Git什么时候我们已经准备好 提交？（put our commits）所以让我们把本地仓库的改动放到我们原来的repo。

我们远程仓库的名字是起源，默认本地分支的名字是主分支master。“-u”告诉git记住指令的参数，这样下次可以直接输入“git push”就可以正常运行了。

# 1.12 Pulling Remotely

Let's pretend some time has passed. We've invited other people to our GitHub project who have pulled your changes, made their own commits, and pushed them.

We can check for changes on our GitHub repository and pull down any new changes by running:

git pull origin master

假如一段时间过去了，我们邀请过其他人登录我们的Github项目，他们已经pull了我们的改动，做出了他们的提交并push到仓库中。我们可以检查我们Github仓库的变化也可以更新任何的改动通过执行下面这个指令。

# 1.13 Differences

Uh oh, looks like there have been some additions and changes to the octocat family. Let's take a look at what is different from our last commit by using the git diff command.

In this case we want the diff of our most recent commit, which we can refer to using the HEAD pointer.

git diff HEAD

看起来octocat发生了一些改动。让我们看看现在与上一次我们使用指令“git diff”有什么不同。这种情况下我们可以通过使用“HEAD”来指出我们大多数最近的提交。

# 1.14 Staged Differences

Another great use for diff is looking at changes within files that have already been staged. Remember, staged files are files we have told git that are ready to be committed.

Let's use git add to stage octofamily/octodog.txt, which I just added to the family for you.

git add octofamily/octodog.txt

另一个好的用法就是查看已经暂存的文件内的改动。记住，暂存文件是我们已经告诉git准备好提交的了。让我们使用这个指令“git add to stage octofamily/octodog.txt”，那个family我刚刚才添加的。

# 1.15 Staged Differences (cont'd)

Good, now go ahead and run git diff with the --staged option to see the changes you just staged. You should see thatoctodog.txt was created.

git diff --staged

现在接着运行“git diff”去查看你更改暂存的改动，你应该可与看到文件“thatocotdog.txt”被创建了。

# 1.16 Resetting the Stage

So now that octodog is part of the family, octocat is all depressed. Since we love octocat more than octodog, we'll turn his frown around by removing octodog.txt.

You can unstage files by using the git reset command. Go ahead and remove octofamily/octodog.txt.

git reset octofamily/octodog.txt

所以现在ocotdog是family的一部分，octocat全部depressed？由于我们喜欢octocat多过octodog，我们将删除octodog.txt。

你可以通过指令“git reset”取消暂存文件（回溯历史）。

# 1.17 Undo

git reset did a great job of unstaging octodog.txt, but you'll notice that he's still there. He's just not staged anymore. It would be great if we could go back to how things were before octodog came around and ruined the party.

Files can be changed back to how they were at the last commit by using the command: git checkout -- <target>. Go ahead and get rid of all the changes since the last commit for octocat.txt

git checkout octocat.txt

Git reset 可以很好的删除octodog.txt但是你会发现他还在那里。他只是不在暂存了。如果我们可以在octodog访问之前回到以前的东西那将是很棒的事情。

文件可以被恢复到他们上一次提交的时候，通过指令“git checkout”实现。

# 1.18 Branching Out

When developers are working on a feature or bug they'll often create a copy (aka. branch) of their code they can make separate commits to. Then when they're done they can merge this branch back into their main master branch.

We want to remove all these pesky octocats, so let's create a branch called clean\_up, where we'll do all the work:

git branch clean\_up

当开发者在使用一个功能或者修改一个bug的时候，他们往往会复制一个他们的代码（分支）这样就可以分开提交操作了。当他们完成工作后悔合并这个分支到他们的主分支上。我们可以删除所有这些讨厌的octocats，所以让我们建立一个名为“clean\_up”的分支，在这里可以完成我们所有的工作。

# 1.19 Switching Branches

Great! Now if you type git branch you'll see two local branches: a main branch named master and your new branch namedclean\_up.

You can switch branches using the git checkout <branch> command. Try it now to switch to the clean\_up branch:

git checkout clean\_up

现在如果你输入“git branch”可以看到两个本地分支，一个是主分支master，另一个是你的新分支clean\_up。

你可以切换分支，通过指令“git checkout”。

# 1.20 Removing All The Things

Ok, so you're in the clean\_up branch. You can finally remove all those pesky octocats by using the git rm command which will not only remove the actual files from disk, but will also stage the removal of the files for us.

You're going to want to use a wildcard again to get all the octocats in one sweep, go ahead and run:

git rm ‘\*.txt’

现在你处于clean\_up分支。你可以通过指令“git rm”最终删除所有的那些讨厌的octocats ，不止是硬盘中实际的文件还包括我们暂存中的文件。你将要再次用到通配符来清扫所有的octocats。

# 1.21 Commiting Branch Changes

Now that you've removed all the cats you'll need to commit your changes.

Feel free to run git status to check the changes you're about to commit.

git commit -m “remove all the cats”

现在你已经删除了所有的cats你将要提交你的改动。试试运行“git status”来检查你将要提交的改动。

# 1.22 Switching Back to master

Great, you're almost finished with the cat... er the bug fix, you just need to switch back to the master branch so you can copy (ormerge) your changes from the clean\_up branch back into the master branch.

Go ahead and checkout the master branch:

git checkout master

你已经几乎完成对cat的操作了。你需要切换回你的主分支这样你才能从clean\_up分支复制你的改动到主分支。

# 1.23 Preparing to Merge

Alrighty, the moment has come when you have to merge your changes from the clean\_up branch into the master branch. Take a deep breath, it's not that scary.

We're already on the master branch, so we just need to tell Git to merge the clean\_up branch into it:

git merge clean\_up

好的，到了从clean\_up分支合并你的改动到主分支的时候了。深呼吸这并不可怕。我们已经在主分支了，所以只需要告诉git合并clean\_up分支进来就可以了。

# 1.24 Keeping Things Clean

Congratulations! You just accomplished your first successful bugfix and merge. All that's left to do is clean up after yourself. Since you're done with the clean\_up branch you don't need it anymore.

You can use git branch -d <branch name> to delete a branch. Go ahead and delete the clean\_up branch now:

git branch -d clean\_up

恭喜你刚刚成功地完成了你的第一次分离、合并分支。接下来需要做的就是清理clean\_up分支，因为你刚刚使用了但现在不需要了。你可以使用指令“branch -d”加上分支名来删除一个分支。

# 1.25 The Final Push

Here we are, at the last step. I'm proud that you've made it this far, and it's been great learning Git with you. All that's left for you to do now is to push everything you've been working on to your remote repository, and you're done!

git push

最后一步，需要做的就是将你刚刚工作的东西都push推送到你的远程仓库中。