1.git允许一群人同时在同一个文件（通常为代码）上工作，而不用紧跟着彼此工作，它是一个分布式版本控制系统。

下面我们的终端提示符是目前在一个我们决定命名为“octobx”的目录。为了初始化一个Git仓库，我们输入以下命令：

git init 初始化仓库

1. 检查git的状态

正如git告诉我们的一样，我们的“octobx”目录现在在/.git/下有一个空的仓库。当Git操作室仓库是一个隐藏目录。

接下来,让我们输入git status命令查看我们的项目的当前状态：

git status 查看仓库状态

1. 增加和提交

我为你在octobx仓库下建立了一个叫做octocat.txt的文件（你可以在浏览器的下面看到）

你可以再运行git status命令观察仓库的状态如何改变：

git status

1. 增加改变

看起来我们的git仓库在正确地工作，注意Git说octocat.txt是无路径的，这说明Git知道octocat.txt是一个新文件。

为了告诉git开始跟踪文件octocat.txt的更改，我们首先需要通过使用命令 git add将它添加到暂存区域：

git add octocat.txt

1. 检查改变

现在git正在跟踪我们的octocat.txt文件，让我们再运行git status看看我们所在的位置。

1. 提交

注意git说提交更改，这里列出的文件在暂存区域，它们还没有在我们的仓库中。在我们将它们存入仓库之前我们可以从暂存区域添加或删除文件。

为了存储暂存区域的更改我们运行commit命令加上一条描述我们所做的更改的信息。现在让我们实现这个通过输入：

git commit -m "Add cute octocat story"

1. 增加所有改变

如果你想添加许多相同类型的文件，你还可以使用通配符。注意我已经添加了一组后缀为.txt 的文件到你的目录下。

我把一些文件放入命名为"octofamily" 的目录和一些其他octobx目录的根目录。幸运的是，我们可以一次性增加所有的文件通过在git add命令后使用通配符。不要忘记使用引号。

1. 提交所有更改

你已经把所有的文本文件都添加到了暂存区。运行 git status看看你将要提交什么样的更改。

如果看起来不错，继续下去运行：

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

1. 历史

这样我们就做了一些提交，现在让我们浏览它们看看我们做了哪些更改。

对我们来说幸运的是，有一个命令git log。我们可以认为Git的日志是一个记得到目前为止我们提交的所有更改的日志，查看提交的更改的顺序，现在试着运行它：

git log

1. 远端仓库

我们已经在<https://github.com/try-git/try_git.git>为你创建了一个新的空GitHub库以便你使用Try Git。为了将我们的本地repo文件放入GitHub服务器，我们需要增加一个远端服务器。

这个命令需要一个远端名字和存储库URL，在<https://github.com/try-git/try_git.git>你可以看实例。

继续下去然后运行git remote add以及后面的选择，如下：

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

1. 推送至远程

push命令告诉git当我们准备好的时候去哪里找到我们的提交，我们已经准备好了。接下来让我们把我们的本地更改放入起始repo文件（在GitHub上）。

我们的远程名字是起始的，默认的本地分支名字是master。-u告诉git记住参数，这样下次我们可以简单运行git push，因为git会知道该执行什么。继续下去然后执行：

git push -u origin master

1. 连接远程

让我们假设过去一段时间了。我们已经邀请了一些人到我们的GitHub项目，他们可以查看你的更改，做他们的提交，合并它们。

我们可以在我们的GitHub仓库查看更改以及下拉新的变化通过运行：

git pull origin master

1. 差异

看起来在octocat中有一些增加和更改，让我们看看与我们最后一次提交有什么区别通过使用 git diff命令。

在这种情况下我们想通过参考使用HEAD指针看到我们最近提交的差异。

git diff HEAD

1. 阶段差异

Diff另一个很大的作用是查看那些已经存储的文件内部的差异。记住，已经存储的文件是我们告诉git那些准备提交的文件。

让我们使用git add分阶段实现octofamily/octodog.txt，这是我刚才为你的家增加的。

git add octofamily/octodog.txt

1. 阶段差异（网际网路企业模式）

现在继续下去运行git diff，后面加上--staged选项看看你存储的更改。你应该看到octodog.txt被建立了。

git diff --staged

1. 重置状态

所以现在octodog是家的一部分，octocat是未定义的，正如我们爱octocat比爱octodog多，我们将删除octodog.txt重新定义octocat。

你可以重定义文件通过使用git reset命令。继续下去然后删除octofamily/octodog.txt：

git reset octofamily/octodog.txt

1. 撤销

git reset做了一个重定义octodog的伟大工作，但你会发现它仍然存在。它只是不再定义。如果我们能够回到octodog建立之前然后彻底销毁它，这将是很好的。

文件可以改回最后一次提交之前它们的样子通过使用命令：git checkout -- <target>。继续下去然后撤销octocat.txt自上次提交的所有更改。

git checkout -- octocat.txt

1. 拓展

当开发人员正在测试一项特性或更改错误的时候。他们经常会为代码建立一个副本（分支），以便做单独的提交。当他们做完后可以合并分支到他们的主分支上。

我们想要删除所有这些讨厌的octocats,所以让我们创建一个名为clean\_up的分支,我们会在此之上做所有的工作：

git branch clean\_up

1. 切换分支

现在如果你输入git branch你将会看到两个本地分支：一个叫做master的主要分支和一个叫做clean\_up的新分支。

你可以通过命令git checkout <branch>切换分支，现在试一下切换到clean\_up分支：

git checkout clean\_up

1. 删除所有东西

现在你正处clean\_up分支中，你终于可以删除所有那些讨厌的octocats通过使用git rm命令，它不仅会从磁盘删除实际的文件,但也将阶段文件删除。你要想再次使用通配符来得到所有分支中的octocats继续运行:

git rm '\*.txt'

21.提交分支更改

现在你已经删除了所有的cats，所以你需要提交你的更改。

运行git status检查你将要提交什么样的更改。

git commit -m "Remove all the cats"

1. 切换到master分支

你几乎完成了所有的cat，或者说是bug修复，你只需要切换回master分支,这样你就可以从clean\_up复制(或合并)更改分支回master分支。

继续下去然后检查master分支：

git checkout master

1. 准备合并

接下来你不得不你的更改从clean\_up分支合并到master分支，深呼吸，这并不可怕。

我们已经在master分支上了，接下来我们只需要告诉git合并clean\_up 到里面：

git merge clean\_up

1. 保持事情清洁

你你刚刚完成你的第一个成功的bug修复和合并。剩下的要做的就是清理你完成的后的东西。既然你完成了clean\_up分支你不就需要它了。

你可以使用 git branch -d <branch name>去删除一个分支。继续下去然后现在删除clean\_up分支：

git branch -d clean\_up

1. 最后一步

这是最后一步，我很自豪你做到了这一步，这是对git很好的一个学习，剩下的给你做的是驱动所有你一直在做的远程存储库,和你想要做的！

git push