# 简易教程：

文件出现在“Untracked files”下面。未跟踪的文件意味着Git在之前的快照（提交）中没有这些文件；Git 不会自动将之纳入跟踪范围，除非你明明白白地告诉它“我需要跟踪该文件”，因而不用担心把临时文件什么的也归入版本管理。

出现在 “Changes not staged for commit” 这行下面，说明已跟踪文件的内容发生了变化，但还没有放到暂存区。要暂存这次更新，以上两种情况需要运行 git add 命令

## git add 文件或路径:

可以用来开始跟踪新文件、或把已跟踪的文件放到暂存区、或用于合并时把有冲突的文件标记为已解决状态等。其实 git add 的潜台词就是把目标文件快照放入暂存区域，也就是 add file into staged area

只要在 “Changes to be committed” 这行下面的，就说明是已暂存状态。如果此时提交（commit），那么该文件此时此刻的版本将被留存在历史记录中。

## git commit: 提交（要输入本次提交的说明）

，尽管使用暂存区域的方式可以精心准备要提交的细节，但有时候这么做略显繁琐。Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤

## git push origin master

把本地分支master中已提交的文件都push到远程仓库origin上。是git push origin master:master的省略形式。

origin指定了你要push到哪个remote

master:master 其实是一个“refspec”，正常的“refspec”的形式为”<src>:<dst>”，冒号前表示local branch的名字，冒号后表示remote repository下 branch的名字。注意，如果你省略了<dst>，git就认为你想push到remote repository下和local branch相同名字的branch。听起来有点拗口，再解释下，就是把本地branch指向的commit push到remote repository下的branch，比如

$git push origin master:master (在local repository中找到名字为master的branch，使用它去更新remote repository下名字为master的branch，如果remote repository下不存在名字是master的branch，那么新建一个)

$git push origin master （省略了<dst>，等价于“git push origin master:master”）

$git push origin master:refs/for/mybranch (在local repository中找到名字为master的branch，用他去更新remote repository下面名字为mybranch的branch)

$git push origin HEAD:refs/for/mybranch （HEAD指向当前工作的branch，master不一定指向当前工作的branch，所以我觉得用HEAD还比master好些）

$git push origin :mybranch （再origin repository里面查找mybranch，删除它。用一个空的去更新它，就相当于删除了）

## git clone [git://github.com/lyz3036/Main-Repository.git](https://github.com/lyz3036/Main-Repository.git) dirName 或：

git clone <https://github.com/lyz3036/Main-Repository.git> dirName

本地没有 repository 时，将远程 repository 整个下载过来，放在dirName文件夹。

## git pull

在clone后出现的文件夹中pull，可以获得最新代码，是本地有 repository 时，将远程 repository 里新的 commit 数据(如有的话)下载过来，并且与本地代码merge。

## git branch master

创建master本地分支

## git branch -a

查看远程分支  
我在mxnet根目录下运行以上命令：

~/mxnet$ git branch -a

\* master

remotes/origin/HEAD -> origin/master

remotes/origin/master

remotes/origin/nnvm

remotes/origin/piiswrong-patch-1

remotes/origin/v0.9rc1

可以看到，我们现在在master分支下

## git branch

创建本地分支

## git checkout master

切换到本地master分支

## git checkout -b localBranch origin/remoteBranch

新建并切换到本地新分支，并与远端分支关联，-b表示创建一个分支

git checkout -b [本地分支名] [远程名]/[远程分支名]

跟踪远程分支

从远程分支 checkout 出来的本地分支，称为 跟踪分支 (tracking branch)。跟踪分支是一种和某个远程分支有直接联系的本地分支。在跟踪分支里输入 git push，Git 会自行推断应该向哪个服务器的哪个分支推送数据。同样，在这些分支里运行 git pull 会获取所有远程索引，并把它们的数据都合并到本地分支中来。

在克隆仓库时，Git 通常会自动创建一个名为 master 的本地分支来跟踪远程分支 origin/master。这正是 git push 和 git pull 一开始就能正常工作的原因。当然，你可以随心所欲地设定为其它跟踪分支，比如远程 origin 上除了 master 之外的其它分支。刚才我们已经看到了这样的一个例子：git checkout -b [本地分支名] [远程名]/[远程分支名]

## 添加origin远程：

git remote add origin <https://github.com/lyz3036/Main-Repository.git>

## 删除当前origin远程

git remote rm origin

## 删除分支

git branch -d Chapater8 可以删除本地分支

git push origin --delete Chapater6 可以删除远程分支Chapater6 （分支名只写origin后的就可以）

或：

删除远程分支

如果不再需要某个远程分支了，比如搞定了某个特性并把它合并进了远程的 master 分支（或任何其他存放稳定代码的分支），可以用这个非常无厘头的语法来删除它：git push [远程名] :[分支名]。如果想在服务器上删除 serverfix 分支，运行下面的命令：

$ git push origin :serverfix

To git@github.com:schacon/simplegit.git

- [deleted] serverfix

咚！服务器上的分支没了。你最好特别留心这一页，因为你一定会用到那个命令，而且你很可能会忘掉它的语法。有种方便记忆这条命令的方法：记住我们不久前见过的 git push [远程名] [本地分支]:[远程分支] 语法，如果省略 [本地分支]，那就等于是在说“在这里提取空白然后把它变成[远程分支]”

## 删除远程中的某个文件、文件夹

思想就是先把所有的远程文件pull下来，再把想要删掉的删掉，再add、commit、push到远程。

法1：

git pull origin master

ls

rm –rf taget

git add

git commit

git push origin master

法2：

git pull origin master //将远程仓库中的项目拉下来

ls

git rm –r –cached taget //删除taget文件夹

git commit //提交

git push –u origin master //将本次更改更新到远程

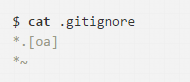
## 本地分支间有区别：

不同本地分支，文件夹中显示的文件不一样，因为在不同的本地分支中提交的文件可能不一样，如下：在localBranch本地分支中有3个文件，切换到master分支中有4个文件，因为提交4个文件时是在master分支中操作的（git commit）



## 忽略某些文件

一般我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。来看一个实际的例子：



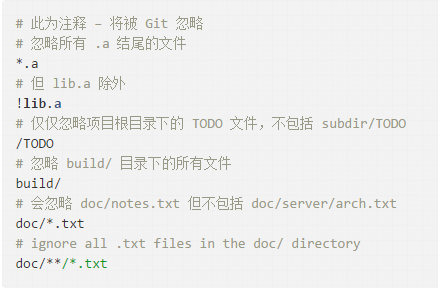
第一行告诉 Git 忽略所有以 .o 或 .a 结尾的文件。一般这类对象文件和存档文件都是编译过程中出现的，我们用不着跟踪它们的版本。第二行告诉 Git 忽略所有以波浪符（~）结尾的文件，许多文本编辑软件（比如 Emacs）都用这样的文件名保存副本。此外，你可能还需要忽略 log，tmp 或者 pid 目录，以及自动生成的文档等等。要养成一开始就设置好 .gitignore 文件的习惯，以免将来误提交这类无用的文件。

文件 .gitignore 的格式规范如下：

* 所有空行或者以注释符号 ＃ 开头的行都会被 Git 忽略。
* 可以使用标准的 glob 模式匹配。
* 匹配模式最后跟反斜杠（/）说明要忽略的是目录。
* 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

所谓的 glob 模式是指 shell 所使用的简化了的正则表达式。星号（\*）匹配零个或多个任意字符；[abc]匹配任何一个列在方括号中的字符（这个例子要么匹配一个 a，要么匹配一个 b，要么匹配一个 c）；问号（?）只匹配一个任意字符；如果在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配（比如 [0-9] 表示匹配所有 0 到 9 的数字）。

我们再看一个 .gitignore 文件的例子：



# 问题解决：

## git 上传文件到仓库上提示：origin does not to be a git repository

最近上传代码到GitHub的时候，当我输入 git push -u origin master的时候，它提示：

origin does not to be a git repository

在网上找到的解决办法：重新输入一次：

git remote add origin https://github.com/lyz3036/Main-Repository.git

然后再输入：

git push -u origin master 就可以提交了

## git 如何正确的提交文件

之所以写这篇文章，是因为昨天发生了一个很重要的事情，那就是我下了一天的代码，在提交的时候没有了。。。因为一些不当的git的操作，我损失了一天的劳动力，并因此付出加班到半夜的代价！因此，在网上查了一下git正确提交文件的步骤，在此做个记录。

一、我的不当操作

1、因为工作一天了，怕在远程代码库的代码被同事修改，我直接传上去会出现问题，所以想要先合并一下，也就是git  pull 下来，但是又觉得直接pull下来，十有八九会直接merge失败，因此我抖机灵一下：

git  checkout  -f

罪恶的根源：

什么是 git checkout ？

答：git checkout本意是切换分支或者创建分支，获取分支的HEAD。或者通过 git checkout master来获取当前master分支的HEAD等。但在使用 git checkout的时候需要注意:

在使用git checkout 时，如果其对应的文件被修改过，那么该修改会被覆盖掉。

git checkout -f 还有一种说法就是放弃本地的修改。所以，这个命令是罪魁祸首。需要慎重使用

**2、**接下来我又执行了  git pull  git push等操作，结果就是全军覆没，写了一天的东西都没了，这是个悲伤的故事。

二、关于正确的提交操作

**一般来说，多人合作开发的时候，都有标准的，就是先提交commit,再pull,最后push**

git add -A #需要添加到版本库中的文件 -A 就代表着所有文件(除开了 .gitignore忽略掉的文件)

git commit -m '本次提交的备注'

git pull #将远程的库和本地的库版本同步一下,因为远程库可能发生了修改(多人合作开发时)

git push

解释一下：

1. commit 是为了告诉 git 我这次提交改了哪些东西,不然你只是改了但是 git 不知道你改了,也就无从判断比较;
2. pull是为了本地 commit 和远程commit 的对比记录,git 是按照文件的行数操作进行对比的,如果同时操作了某文件的同一行那么就会产生冲突,git 也会把这个冲突给标记出来,这个时候就需要先把和你冲突的那个人拉过来问问保留谁的代码,然后在 git add && git commit && git pull 这三连,再次 pull 一次是为了防止再你们协商的时候另一个人给又提交了一版东西,如果真发生了那流程重复一遍,通常没有冲突的时候就直接给你合并了,不会把你的代码给覆盖掉
3. 出现代码覆盖或者丢失的情况:比如A B两人的代码pull 时候的版本都是1,A在本地提交了2,3并且推送到远程了,B 进行修改的时候没有commit 操作,他先自己写了东西,然后 git pull 这个时候 B 本地版本已经到3了,B 在本地版本3的时候改了 A 写过的代码,再进行了git commit && git push 那么在远程版本中就是4,而且 A 的代码被覆盖了,所以说所有人都要先 commit 再 pull,不然真的会覆盖代码的

所以，以后我们都要记住，操作git的时候需要谨慎。也别是如我般的git新手，一定要小心小心再小心。幸亏昨天只是影响我自己的工作，如果是影响整个项目，那罪过就太大了。