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 <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

## 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]

## 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

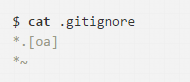
To git@github.com:schacon/simplegit.git

- [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]”

## 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：

