预备知识：

CMD：

快捷键：win+R->cmd->enter

在窗口里调用的命令分为两类，一类是原生的命令，还有一类是各种支持命令行的程序

一般看到一个命令后，如果没有说明文档，你就尝试的在其命令后直接输入 -help 或者 /? 再或者一个随便什么的字符让它报错，一般情况下就会给出这个命令下相应的参数说明。

后面的参数使用方法这样来进行解读：一个中括号里面的是一组参数，括号中间|的是或的意思，比如一个红框包住的部分，你不能即+R又-R，二者择其一！最后的[/S [/D] [/L]]可以多选。 --------------------- 本文来自 xjz729827161 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/xjz729827161/article/details/53513060?utm\_source=copy

gcc and g++分别是gnu（**GNU**是一个[自由](https://baike.baidu.com/item/%E8%87%AA%E7%94%B1)的[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)，其内容软件完全以[GPL](https://baike.baidu.com/item/GPL)方式发布）的c & c++编译器 gcc/g++在执行编译工作的时候，总共需要4步 1.预处理,生成.i的文件[预处理器cpp] 2.将预处理后的文件不转换成汇编语言,生成文件.s[编译器egcs] 3.有汇编变为目标代码(机器代码)生成.o的文件[汇编器as] 4.连接目标代码,生成可执行程序[链接器ld]

一、

 Git 有三种状态，你的文件可能处于其中之一：已提交（committed）、已修改（modified）和已暂存（staged）。

基本的 Git 工作流程如下：

1. 在工作目录中修改文件。
2. 暂存文件，将文件的快照放入暂存区域。
3. 提交更新，找到暂存区域的文件，将快照永久性存储到 Git 仓库目录。

如果 Git 目录中保存着的特定版本文件，就属于已提交状态。 如果作了修改并已放入暂存区域，就属于已暂存状态。 如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

二、

若你使用 Git 时需要获取帮助，有三种方法可以找到 Git 命令的使用手册：

$ git help <verb>

$ git <verb> --help

$ man git-<verb>

例如，要想获得 config 命令的手册，执行

$ git help config

这些命令很棒，因为你随时随地可以使用而无需联网。 如果你觉得手册或者本书的内容还不够用，你可以尝试在 Freenode IRC 服务器（ irc.freenode.net ）的 #git 或 #github 频道寻求帮助。 这些频道经常有上百人在线，他们都精通 Git 并且乐于助人。

三、

有两种取得 Git 项目仓库的方法。 第一种是在现有项目或目录下导入所有文件到 Git 中； 第二种是从一个服务器克隆一个现有的 Git 仓库。

克隆仓库的命令格式是 git clone [url] 。 比如，要克隆 Git 的可链接库 libgit2，可以用下面的命令：

$ git clone https://github.com/libgit2/libgit2

这会在当前目录下创建一个名为 “libgit2” 的目录，并在这个目录下初始化一个 .git 文件夹，从远程仓库拉取下所有数据放入 .git 文件夹，然后从中读取最新版本的文件的拷贝。 如果你进入到这个新建的 libgit2 文件夹，你会发现所有的项目文件已经在里面了，准备就绪等待后续的开发和使用。 如果你想在克隆远程仓库的时候，自定义本地仓库的名字，你可以使用如下命令：

$ git clone https://github.com/libgit2/libgit2 mylibgit

这将执行与上一个命令相同的操作，不过在本地创建的仓库名字变为 mylibgit。

现在，让我们在项目下创建一个新的 README 文件。 如果之前并不存在这个文件，使用 git status 命令，你将看到一个新的未跟踪文件：

$ echo 'My Project' > README

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

README

nothing added to commit but untracked files present (use "git add" to track)

 git add 命令使用文件或目录的路径作为参数；如果参数是目录的路径，该命令将递归地跟踪该目录下的所有文件。

四、

**忽略文件**

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。 在这种情况下，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：

$ cat .gitignore

\*.[oa]

\*~

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。 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 ＃ 开头的行都会被 Git 忽略。
* 可以使用标准的 glob 模式匹配。
* 匹配模式可以以（/）开头防止递归。
* 匹配模式可以以（/）结尾指定目录。
* 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）。 使用两个星号（\*) 表示匹配任意中间目录，比如`a/\*\*/z` 可以匹配 a/z, a/b/z 或 `a/b/c/z`等。

我们再看一个 .gitignore 文件的例子：

# no .a files

\*.a

# but do track lib.a, even though you're ignoring .a files above

!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO

/TODO

# ignore all files in the build/ directory

build/

# ignore doc/notes.txt, but not doc/server/arch.txt

doc/\*.txt

# ignore all .pdf files in the doc/ directory

doc/\*\*/\*.pdf

五、git diff

尽管 git status 已经通过在相应栏下列出文件名的方式回答了这个问题，git diff 将通过文件补丁的格式显示具体哪些行发生了改变。

六、提交

编辑器会显示类似下面的文本信息（本例选用 Vim 的屏显方式展示）：

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

# On branch master

# Changes to be committed:

# new file: README

# modified: CONTRIBUTING.md

#

~

~

~

".git/COMMIT\_EDITMSG" 9L, 283C

可以看到，默认的提交消息包含最后一次运行 git status 的输出，放在注释行里，另外开头还有一空行，供你输入提交说明。 你完全可以去掉这些注释行，不过留着也没关系，多少能帮你回想起这次更新的内容有哪些。 (如果想要更详细的对修改了哪些内容的提示，可以用 -v 选项，这会将你所做的改变的 diff 输出放到编辑器中从而使你知道本次提交具体做了哪些修改。） 退出编辑器时，Git 会丢掉注释行，用你输入提交附带信息生成一次提交。

另外，你也可以在 commit 命令后添加 -m 选项，将提交信息与命令放在同一行，如下所示：

$ git commit -m "Story 182: Fix benchmarks for speed"

[master 463dc4f] Story 182: Fix benchmarks for speed

2 files changed, 2 insertions(+)

create mode 100644 README

好，现在你已经创建了第一个提交！ 可以看到，提交后它会告诉你，当前是在哪个分支（master）提交的，本次提交的完整 SHA-1 校验和是什么（463dc4f），以及在本次提交中，有多少文件修订过，多少行添加和删改过。

请记住，提交时记录的是放在暂存区域的快照。 任何还未暂存的仍然保持已修改状态，可以在下次提交时纳入版本管理。 每一次运行提交操作，都是对你项目作一次快照，以后可以回到这个状态，或者进行比较。

### 七、跳过使用暂存区域

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 status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -a -m 'added new benchmarks'

[master 83e38c7] added new benchmarks

1 file changed, 5 insertions(+), 0 deletions(-)

看到了吗？提交之前不再需要 git add 文件“CONTRIBUTING.md”了。