外壳脚本

Shell 脚本是一个很大的话题——毕竟 shell 是一种完整的编程语言。在本次活动的视频中，您只看到了非常简短的介绍。

编译助手练习

在名为（用于构建）的文件中编写一个 shell 脚本 `b`，该脚本执行以下操作：

- 您的脚本应该在任何与 Bourne 兼容的 shell 下运行（例如不仅仅是 `bash`），并且应该编写它以便您可以使用 `./b`。
- `./b compile NAME` 应该编译给定名称的文件，因此例如 `./b compile hello` 应该运行 `gcc -Wall -std=c11 -g hello.c -o hello`。
- 但是，您的脚本应该接受 `./b compile hello` 和 `./b compile hello.c` 作为输入，并在两种情况下执行相同的操作，即编译 `hello.c`。在这两种情况下，gcc 的输出文件都应该被称为 `hello`。
- 如果您作为参数提供的源文件不存在（`.c` 如有必要则添加），则脚本应打印一条错误消息并返回非零退出状态 -不调用 C 编译器。
- `./b run NAME` 应该运行该程序，假设它存在于当前文件夹中，因此 和 都 `./b run hello` 应该 `./b run hello.c` 运行 `./hello`。如果它不存在，则再次打印错误消息并以非零状态退出，不要尝试运行该程序。
- `./b build NAME` 应该首先编译C源文件，然后如果编译成功则应该运行该程序。如果编译失败，则不应尝试运行该程序。
- 如果您 `./b` 在不带任何参数的情况下调用，或者 `./b COMMAND` 使用编译、运行或构建以外的命令进行调用，则它应该打印一些有关如何使用它的信息。如果您调用 `./b compile` 或另一个根本没有文件名的命令，则脚本应打印一条错误消息并以非零退出状态退出。

现在，您在开发 C 程序时拥有了一个有用的工具。当然，您可以添加自己的其他功能，例如 `debug` 编译文件并在 `gdb`。

如果您已经了解 makefile，您可能想知道为什么我们不只用于 `make` 此目的。你是对的，但这只是一个 shell 脚本练习。我们很快就会了解 makefile。

严格模式

一些编程语言有一个可选的*严格模式*，它将某些结构视为人们经常错误地执行的错误。它在本质上与 C 语言类似 `-Werror`，将所有警告视为错误。本页建议在 shell 脚本顶部附近使用以下行： `set`

`-euo pipefail`。（它还讨论了 IFS 来改进带空格的字符串处理，但这是一个单独的问题。）

如果您开始编写 shell 脚本，您可能想自己使用这些。 `set` 是一个 shell 内部命令，用于设置控制命令运行方式的 shell 标志。

- `set -e` 如果任何命令失败，则使整个脚本退出。这样，如果你想运行一系列命令，你可以将它们放在 `set -e` 顶部带有 的脚本中，只要所有命令都成功（返回 0），shell 就会继续执行；如果任何命令返回非零，它将停止进一步运行。这就像把 `|| exit $?` 每条命令放在最后。
- `set -u` 意味着引用未定义的变量是错误的。出于多种原因，这是一个很好的做法。
- `set -o pipefail` 改变管道的工作方式：通常，管道的返回值是管道中*最后一个命令的返回值*。使用该 `pipefail` 选项，如果管道中的任何命令失败（非零返回），则管道返回该命令的退出代码。

关于 的一些注意事项 `set -u`： 如果您编写类似 `rm -rf $FOLDER/` 和 `$FOLDER` 未设置的内容，那么您不会意外地删除整个系统！当然，大多数实现将拒绝在没有该选项的情况下 `rm` 删除，并且您不应该首先使用尾随斜杠。Linux 版 Steam 的 Beta 版本中存在一个错误，它试图删除文件夹中的所有文件（这解释了斜线），但在某些情况下该变量被设置为*空字符串*，这无法防止。这是一个安装程序脚本，因此它以 root 身份运行，这使事情变得更糟。 `/ --no-preserve-root rm -rf "$STEAMROOT/"* -u`

练习：考虑 shell 脚本中的一个示例，其中的 `pipefail` 区别在于，即使前一个命令失败，管道中的最后一个命令也可能成功。作为一个反例，即使文件不存在， `cat FILE | grep STRING` 也会失败 `pipefail`，因为 grep 会立即在标准输入上获取文件结尾。