动机

作为计算机科学家，我们都知道计算机最擅长帮助我们完成重复性的工作。
但是我们却常常忘记这一点也适用于我们使用计算机的方式，而不仅仅是利用计算机程序去帮我们求解问题。
在从事与计算机相关的工作时，我们有很多触手可及的工具可以帮助我们更高效的解决问题。
但是我们中的大多数人实际上只利用了这些工具中的很少一部分，我们常常只是死记硬背一些如咒语般的命令，
或是当我们卡住的时候，盲目地从网上复制粘贴一些命令。

本课程意在帮你解决这一问题。

我们希望教会您如何挖掘现有工具的潜力，并向您介绍一些新的工具。也许我们还可以促使您想要去探索（甚至是去开发）更多的工具。
我们认为这是大多数计算机科学相关课程中缺少的重要一环。

课程结构

本课程包含 11 个时长在一小时左右的讲座，每一个讲座都会关注一个
特定的主题。尽管这些讲座之间基本上是各自独立的，但随着课程的进行，我们会假定您已经掌握了之前的内容。
每个讲座都有在线笔记供查阅，但是课上的很多内容并不会包含在笔记中。因此我们也会把课程录制下来发布到互联网上供大家观看学习。

我们希望能在这 11 个一小时讲座中涵盖大部分必须的内容，因此课程的信息密度是相当大的。为了能帮助您以自己的节奏来掌握讲座内容，每次课程都包含一组练习来帮助您掌握本节课的重点。
课后我们会安排答疑的时间来回答您的问题。如果您参加的是在线课程，可以发送邮件到
missing-semester@mit.edu 来联系我们。

由于时长的限制，我们不可能达到那些专门课程一样的细致程度，我们会适时地将您介绍一些优秀的资源，帮助您深入的理解相关的工具或主题。
但是如果您还有一些特别关注的话题，也请联系我们。

主题 1: The Shell

shell 是什么?

如今的计算机有着多种多样的交互接口让我们可以进行指令的的输入，从炫酷的图像用户界面（GUI），语音输入甚至是 AR/VR 都已经无处不在。
这些交互接口可以覆盖 80% 的使用场景，但是它们也从根本上限制了您的操作方式——你不能点击一个不存在的按钮或者是用语音输入一个还没有被录入的指令。
为了充分利用计算机的能力，我们不得不回到最根本的方式，使用文字接口：Shell

几乎所有您能够接触到的平台都支持某种形式的 shell，有些甚至还提供了多种 shell 供您选择。虽然它们之间有些细节上的差异，但是其核心功能都是一样的：它允许你执行程序，输入并获取某种半结构化的输出。

本节课我们会使用 Bourne Again SHell, 简称 "bash" 。
这是被最广泛使用的一种 shell，它的语法和其他的 shell 都是类似的。打开 shell *提示符*（您输入指令的地方），您首先需要打开 *终端* 。您的设备通常都已经内置了终端，或者您也可以安装一个，非常简单。

使用 shell

当您打开终端时，您会看到一个提示符，它看起来一般是这个样子的：

```
missing:~$
```

这是 shell 最主要的文本接口。它告诉你，你的主机名是 `missing` 并且您当前的工作目录（"current working directory"）或者说您当前所在的位置是 `~` (表示 "home")。 `$` 符号表示您现在的身份不是 root 用户（稍后会介绍）。在这个提示符中，您可以输入 *命令* ，命令最终会被 shell 解析。最简单的命令是执行一个程序：

```
missing:~$ date
Fri 10 Jan 2020 11:49:31 AM EST
missing:~$
```

这里，我们执行了 `date` 这个程序，不出意料地，它打印出了当前的日期和时间。然后，shell 等待我们输入其他命令。我们可以在执行命令的同时向程序传递 *参数* ：

```
missing:~$ echo hello
hello
```

上例中，我们让 shell 执行 `echo` ，同时指定参数 `hello`。`echo` 程序将该参数打印出来。
shell 基于空格分割命令并进行解析，然后执行第一个单词代表的程序，并将后续的单词作为程序可以访问的参数。如果您希望传递的参数中包含空格（例如一个名为 My Photos 的文件夹），您要么用使用单引号，双引号将其包裹起来，要么使用转义符号 `\` 进行处理（`My\ Photos`）。

但是，shell 是如何知道去哪里寻找 `date` 或 `echo` 的呢？其实，类似于 Python 或 Ruby，shell 是一个编程环境，所以它具备变量、条件、循环和函数（下一课进行讲解）。当你在 shell 中执行命令时，您实际上是在执行一段 shell 可以解释执行的简短代码。如果你要求 shell 执行某个指令，但是该指令并不是 shell 所了解的编程关键字，那么它会去咨询 *环境变量* `$PATH`，它会列出当 shell 接到某条指令时，进行程序搜索的路径：

```
missing:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
missing:~$ which echo
/bin/echo
missing:~$ /bin/echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

当我们执行 `echo` 命令时，shell 了解到需要执行 `echo` 这个程序，随后它便会在 `$PATH` 中搜索由 `:` 所分割的一系列目录，基于名字搜索该程序。当找到该程序时便执行（假定该文件是 *可执行程序*，后续课程将详细讲解）。确定某个程序名代表的是哪个具体的程序，可以使用
`which` 程序。我们也可以绕过 `$PATH`，通过直接指定需要执行的程序的路径来执行该程序

在 shell 中导航

shell 中的路径是一组被分割的目录，在 Linux 和 macOS 上使用 `/` 分割，而在 Windows 上是 `\`。路径 `/` 代表的是系统的根目录，所有的文件夹都包括在这个路径之下，在 Windows 上每个盘都有一个根目录（例如：
`C:\`）。 我们假设您在学习本课程时使用的是 Linux 文件系统。如果某个路径以 `/` 开头，那么它是一个 *绝对路径*，其他的都是 *相对路径* 。相对路径是指相对于当前工作目录的路径，当前工作目录可以使用 `pwd` 命令来获取。此外，切换目录需要使用 `cd` 命令。在路径中，`.` 表示的是当前目录，而 `..` 表示上级目录：

```
missing:~$ pwd
/home/missing
```

```
missing:~$ cd /home
missing:/home$ pwd
/home
missing:/home$ cd ..
missing:/$ pwd
/
missing:/$ cd ./home
missing:/home$ pwd
/home
missing:/home$ cd missing
missing:~$ pwd
/home/missing
missing:~$ ../../bin/echo hello
hello
```

注意，shell 会实时显示当前的路径信息。您可以通过配置 shell 提示符来显示各种有用的信息，这一内容我们会在后面的课程中进行讨论。

一般来说，当我们运行一个程序时，如果我们没有指定路径，则该程序会在当前目录下执行。例如，我们常常会搜索文件，并在需要时创建文件。

为了查看指定目录下包含哪些文件，我们使用 `ls` 命令：

```
missing:~$ ls
missing:~$ cd ..
missing:/home$ ls
missing
missing:/home$ cd ..
missing:/$ ls
bin
boot
dev
etc
home
...
```

除非我们利用第一个参数指定目录，否则 `ls` 会打印当前目录下的文件。大多数的命令接受标记和选项（带有值的标记），它们以 `-` 开头，并可以改变程序的行为。通常，在执行程序时使用 `-h` 或 `--help` 标记可以打印帮助信息，以便了解有哪些可用的标记或选项。例如，`ls --help` 的输出如下：

```
  -l                         use a long listing format
```

```
missing:~$ ls -l /home
drwxr-xr-x 1 missing  users  4096 Jun 15  2019 missing
```

这个参数可以更加详细地列出目录下文件或文件夹的信息。首先，本行第一个字符 `d` 表示 `missing` 是一个目录。然后接下来的九个字符，每三个字符构成一组。（`rwx`）. 它们分别代表了文件所有者（`missing`），用户组（`users`） 以及其他所有人具有的权限。其中 `-` 表示该用户不具备相应的权限。从上面的信息来看，只有文件所有者可以修改（`w`），`missing` 文件夹 （例如，添加或删除文件夹中的文件）。为了进入某个文件夹，用户需要具备该文件夹以及其父文件夹的“搜索”权限（以“可执行”：`x`）权

限表示。为了列出它的包含的内容，用户必须对该文件夹具备读权限（`r`）。对于文件来说，权限的意义也是类似的。
注意，`/bin` 目录下的程序在最后一组，即表示所有人的用户组中，均包含 `x` 权限，也就是说任何人都可以执行这些程序。

在这个阶段，还有几个趁手的命令是您需要掌握的，例如 `mv`（用于重命名或移动文件）、`cp`（拷贝文件）以及 `mkdir`（新建文件夹）。

如果您想要知道关于程序参数、输入输出的信息，亦或是想要了解它们的工作方式，请试试 `man` 这个程序。它会接受一个程序名作为参数，然后将它的文档（用户手册）展现给您。注意，使用 `q` 可以退出该程序。

```
missing:~$ man ls
```

在程序间创建连接

在 shell 中，程序有两个主要的“流”：它们的输入流和输出流。
当程序尝试读取信息时，它们会从输入流中进行读取，当程序打印信息时，它们会将信息输出到输出流中。
通常，一个程序的输入输出流都是您的终端。也就是，您的键盘作为输入，显示器作为输出。
但是，我们也可以重定向这些流！

最简单的重定向是 `< file` 和 `> file`。这两个命令可以将程序的输入输出流分别重定向到文件：

```
missing:~$ echo hello > hello.txt
missing:~$ cat hello.txt
hello
missing:~$ cat < hello.txt
hello
missing:~$ cat < hello.txt > hello2.txt
missing:~$ cat hello2.txt
hello
```

您还可以使用 `>>` 来向一个文件追加内容。使用管道（ *pipes* ），我们能够更好的利用文件重定向。
`|` 操作符允许我们将一个程序的输出和另外一个程序的输入连接起来：

```
missing:~$ ls -l / | tail -n1
drwxr-xr-x 1 root  root  4096 Jun 20  2019 var
missing:~$ curl --head --silent google.com | grep --ignore-case content-length | cut --
delimiter=' ' -f2
219
```

我们会在数据清理一章中更加详细的探讨如何更好的利用管道。

一个功能全面又强大的工具

对于大多数的类 Unix 系统，有一类用户是非常特殊的，那就是：根用户（root user）。
您应该已经注意到了，在上面的输出结果中，根用户几乎不受任何限制，他可以创建、读取、更新和删除系统中的任何文件。
通常在我们并不会以根用户的身份直接登录系统，因为这样可能会因为某些错误的操作而破坏系统。
取而代之的是我们会在需要的时候使用 `sudo` 命令。顾名思义，它的作用是让您可以以 su（super user 或 root 的简写）的身份执行一些操作。

当您遇到拒绝访问（permission denied）的错误时，通常是因为此时您必须是根用户才能操作。然而，请再次确认您是真的要执行此操作。

有一件事情是您必须作为根用户才能做的，那就是向 `sysfs` 文件写入内容。系统被挂载在 `/sys` 下，`sysfs` 文件则暴露了一些内核（kernel）参数。
因此，您不需要借助任何专用的工具，就可以轻松地在运行期间配置系统内核。**注意 Windows 和 macOS 没有这个文件**

例如，您笔记本电脑的屏幕亮度写在 `brightness` 文件中，它位于

```
/sys/class/backlight
```

通过将数值写入该文件，我们可以改变屏幕的亮度。现在，蹦到您脑袋里的第一个想法可能是：

```
$ sudo find -L /sys/class/backlight -maxdepth 2 -name '*brightness*'
/sys/class/backlight/thinkpad_screen/brightness
$ cd /sys/class/backlight/thinkpad_screen
$ sudo echo 3 > brightness
An error occurred while redirecting file 'brightness'
open: Permission denied
```

出乎意料的是，我们还是得到了一个错误信息。毕竟，我们已经使用了
`sudo` 命令！关于 shell，有件事我们必须要知道。`|`、`>`、和 `<` 是通过 shell 执行的，而不是被各个程序单独执行。`echo` 等程序并不知道 `|` 的存在，它们只知道从自己的输入输出流中进行读写。
回到上面更改屏幕亮度命令执行的报错，为了能让 `sudo echo` 命令输出的亮度值写入 brightness 文件， *shell* (权限为当前用户) 会先尝试打开 brightness 文件，但此时操作 shell 的不是根（root）用户，所以系统拒绝了这个打开操作，提示无权限。

明白这一点后，我们可以这样操作：

```
$ echo 3 | sudo tee brightness
```

此时打开 `/sys` 文件的是 `tee` 这个程序，并且该程序以 `root` 权限在运行，因此操作可以进行。
这样您就可以在 `/sys` 中愉快地玩耍了，例如修改系统中各种 LED 的状态（路径可能会有所不同）：

```
$ echo 1 | sudo tee /sys/class/leds/input6::scrolllock/brightness
```

接下来.....

学到这里，您掌握的 shell 知识已经可以完成一些基础的任务了。您应该已经可以查找感兴趣的文件并使用大多数程序的基本功能了。
在下一场讲座中，我们会探讨如何利用 shell 及其他工具执行并自动化更复杂的任务。

课后练习

习题解答
本课程中的每节课都包含一系列练习题。有些题目是有明确目的的，另外一些则是开放题，例如“尝试使用 X 和 Y”，我们强烈建议您一定要动手实践，用于尝试这些内容。
此外，我们没有为这些练习题提供答案。如果有任何困难，您可以发送邮件给我们并描述你已经做出的尝试，我们会

设法帮您解答。

1. 本课程需要使用类 Unix shell，例如 Bash 或 ZSH。如果您在 Linux 或者 MacOS 上面完成本课程的练习，则不需要做任何特殊的操作。如果您使用的是 Windows，则您不应该使用 cmd 或是 Powershell；您可以使用 Windows Subsystem for Linux 或者是 Linux 虚拟机。使用 `echo $SHELL` 命令可以查看您的 shell 是否满足要求。如果打印结果为 `/bin/bash` 或 `/usr/bin/zsh` 则是可以的。
2. 在 `/tmp` 下新建一个名为 `missing` 的文件夹。
3. 用 `man` 查看程序 `touch` 的使用手册。
4. 用 `touch` 在 `missing` 文件夹中新建一个叫 `semester` 的文件。
5. 将以下内容一行一行地写入 `semester` 文件：

   ```
   #!/bin/sh
   curl --head --silent https://missing.csail.mit.edu
   ```

   第一行可能有点棘手， `#` 在 Bash 中表示注释，而 `!` 即使被双引号（`"`）包裹也具有特殊的含义。单引号（`'`）则不一样，此处利用这一点解决输入问题。更多信息请参考 Bash quoting 手册
6. 尝试执行这个文件。例如，将该脚本的路径（`./semester`）输入到您的 shell 中并回车。如果程序无法执行，请使用 `ls` 命令来获取信息并理解其不能执行的原因。
7. 查看 `chmod` 的手册(例如，使用 `man chmod` 命令)
8. 使用 `chmod` 命令改变权限，使 `./semester` 能够成功执行，不要使用 `sh semester` 来执行该程序。您的 shell 是如何知晓这个文件需要使用 `sh` 来解析呢？更多信息请参考：shebang
9. 使用 `|` 和 `>` ，将 `semester` 文件输出的最后更改日期信息，写入主目录下的 `last-modified.txt` 的文件中
10. 写一段命令来从 `/sys` 中获取笔记本的电量信息，或者台式机 CPU 的温度。注意：macOS 并没有 sysfs，所以 Mac 用户可以跳过这一题。