# Comp2700 实验 2：Linux 命令和 shell 脚本

在本实验室中，我们将学习 Linux 中一些更高级的概念和命令，继续实验室 1 中的讨论，以及 Linux shell 脚本。Shell 脚本在类 Unix 系统管理中广泛使用，了解它们的工作原理和缺点是理解 Unix 安全性的重要部分。本实验旨在为后续课程中与 Unix/Linux 安全相关的高级讲座和实验提供最基本的背景知识；而不是 Linux 管理或 shell 脚本编程的速成课程。如果您想了解有关 shell 脚本编程的更多信息，本实验手册末尾提供了一些参考资料。

本实验室要求您完成一些练习。这些练习是为了让您理解每节中的概念。实验结束后，您需要完成实验测验，测验结果将发布在 Wattle 课程网站上。测验必须在测验之日起一周内完成。详情将在 Wattle 课程网站上提供。

一个或多个练习标有 "*"，表示在实验过程中不会讨论的可选扩展练习。我们不要求您完成这些练习，但鼓励您完成这些练习。

完成本实验后，学生应能

1. 能够在 shell 上执行基本的输入/输出操作：创建简单文件、写入、读取或追加文件内容、搜索文件内容以及重定向文件的输入/输出操作。
2. 展示对环境变量概念以及如何读写环境变量的理解。
3. 能够执行简单的 shell 命令组合：顺序组合和管道。
4. 编写简单的 shell 脚本，包括基本控制命令：if-then-else、字符串和整数比较以及循环。

## 0.实验室设置

***符号约定：****在下文中，当编写 shell 命令时，我们将使用以符号"$"开头的行来表示它们是 shell 命令。例如*

```
$ ls
```

*表示执行 shell 命令 "ls"（用于列出文件和目录）。命令中不包含符号 $。*

实验室设置与实验室 1 相同，因此我们将继续使用在实验室 1 中设置的实验室虚拟机。我们还将使用在实验室 1 中提供的 **lab1.tar.gz**。我们将在此重复本实验中再次使用的 "lab1 "目录的设置说明：

1. 以用户 admin2700 登录实验室虚拟机。
2. 使用以下命令删除之前实验室中的 lab1 副本（如果有的话）：

   ```
   $ sudo rm -rf /home/alice/lab1
   ```

3. 更改为用户 **alice**：

   ```
   $ su -l alice
   ```

4. 如果尚未下载文件 **lab1.tar.gz**，请下载该文件。

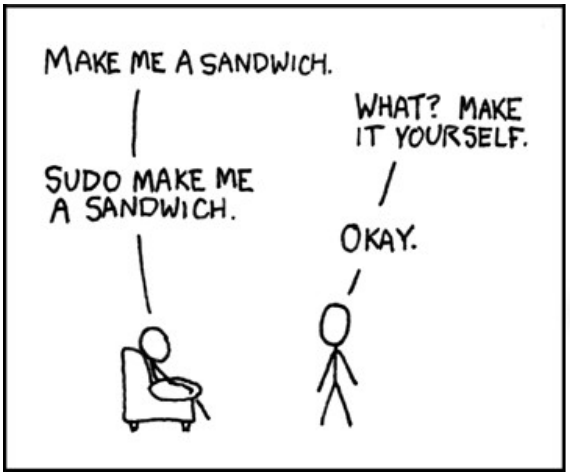
   ```
   $ cd ~/
   $ wget http://users.cecs.anu.edu.au/~tiu/comp2700/lab1.tar.gz
   ```

5. 使用以下命令解压 lab1.tar.gz 中的文件（假设你把文件放在 alice 的主目录下）：

   ```
   $ extract-lab lab1.tar.gz
   ```

   这将在 /home/alice 中创建一个名为 lab1 的目录。

## 关于 sudo 命令的说明




(来源：xkcd.com)

有时，你可能需要执行一些需要升级权限的管理任务，如添加/删除其他用户创建的文件（如上文实验室设置步骤 2 所示）、更改文件所有权、更改可执行文件的某些访问控制策略等。在 Linux 中，只有 "root "账户可以不受限制地访问系统中的所有资源，但不能直接以 root 用户身份连接系统。不过，管理员账户（如实验室中的 "admin2700

VM）可以使用 "sudo "命令将自己的权限升级到相当于 "root "账户的权限。在命令前加上 "sudo " 会使该命令以升级后的权限运行。要在实验室虚拟机中使用 sudo 命令，需要以用户 admin2700 的身份登录。

我们将在 Linux/Unix 安全实验室中再次讨论 sudo 命令的使用。请谨慎使用该命令，仅在需要时使用。

## 1.文件操作

在实验 1 中，我们了解了如何对文件执行简单的操作：复制、删除和显示文件内容。下面我们将学习一些更高级的命令。

### 搜索文件内容

**语法**

***grep [-options] 模式文件***

**Grep** 命令用于搜索与模式（指定为正则表达式）相匹配的文本行，并只输出匹配的行。

例如

```
$ grep Chapter ~/lab1/sample_files/comp2700.txt
```

将显示 comp2700.txt 中包含 "Chapter "字符串的行。

grep 第一个参数中的模式可以是任何*正则表达式*；我们将不在这里讨论它们，但如果你感兴趣，grep 的 man 页面会提供更多详细信息（使用 "man grep "调出 man 页面）。

选项 -n 可用于显示匹配文本的行号。例如

```
$ grep -n Chapter ~/lab1/sample_files/comp2700.txt
```

将产生与前一条命令类似的结果，但输出中的每一行都将以一个数字作为前缀，表示输入文件中匹配文本的行号。

与 `ls'命令一样，[File] 参数可以用模式代替；在这种情况下，将搜索与该模式匹配的所有文件。

选项 -R 可用于递归搜索一个目录（包括子目录中的所有文件）。

**练习 1**. 使用 grep 搜索 /etc/ 目录（包括其所有子目录）中所有文件的内容，查找关键字 "password"（忽略大小写，因此搜索范围应包括包含 "Passwords "的文件）。*请注意，你可能会遇到包含 "权限被拒绝 "的错误信息。你可以暂时忽略它。*

## 查找文件

语法

***文件名***

该命令显示命令***文件名***的完整路径。例如：运行 ***ls*** 会得到"/usr/bin/ls "的答案。也就是说，"ls "命令指的是可执行文件
/usr/bin/ls。

如果您有同名但位于不同目录下的不同可执行文件，而您又想知道在输入命令名时调用的是哪个版本，那么这条命令就非常有用。

***找到***

该命令搜索符合特定模式和其他限制条件的文件。这是一个非常强大的查找文件的命令，有很多选项，我们在这里就不一一介绍了。我们将在以后的实验中看到 "find "的更高级用法。常见的用法如下：

***查找 [路径] -名称 [搜索字符串]***

该命令检查给定路径下的所有文件和子目录，并打印名称中包含搜索字符串的文件名。

例如

```
$ find /home/alice/lab1 -name test.txt
```

搜索名称为 "test.txt "的文件，并打印匹配文件的完整路径。

```
alice@comp2700_lab:~$ find /home/alice -name test.txt
/home/alice/lab1/Q2/test.txt
/home/alice/lab1/test.txt
alice@comp2700_lab:~$
```

搜索字符串可以是一个模式。例如，下面的命令将查找名称以 Q 开头的所有文件/目录

```
$ find /home/alice -name "Q*"
```

注意，模式必须用双引号括起来。

**练习 2.** 使用 "find "命令，查找 /home/alice 中的所有隐藏文件。在 Linux 中，隐藏文件是指名称以点". "开头的文件，例如 `.bashrc`。

### 执行文件

要执行（程序）文件，只需指定文件的完整路径。例如

```
$ /home/alice/lab1/hello
```

执行 lab1 目录中的 hello 程序。

例如，与 Windows 不同，Linux 中的可执行文件不需要任何特定的扩展名（如 "exe "或 "com"）。相反，可执行文件通过与文件相关的 "可执行 "权限位来表示--我们将在有关 Unix 安全性的讲座中详细介绍这一点。

这里也可以使用目录的各种快捷方式，例如，执行上述命令（由 Alice 执行）时，可以使用

```
$ ~/lab1/hello
```

因为 ~ 会展开为 /home/alice。同样，如果当前目录是 /home/alice/lab1/，我们只需键入

```
$ ./hello
```

因为点 (`.') 表示当前目录 /home/alice/lab1。

例如，我们只需键入 "ls "而不是"/usr/bin/ls "就可以执行 ls 文件。这是因为这些命令的程序都在 shell 环境的搜索路径中（稍后我们将讨论环境变量）。

> **练习 3.** 查找与 shell 命令 "cat "相对应的程序的位置，并将其复制到 Alice 的主目录，然后重命名为 "mycat"。运行 "mycat "程序，显示 ~/lab1/ab.txt 的内容。

## 2.输入/输出重定向

### 输入重定向 (<)

输入重定向允许你用文件代替标准输入（即键盘）。例如，我们前面看到的 grep 命令。如果该命令的参数中没有提供文件名，那么它将从标准输入中获取输入，例如，如果输入

```
$ grep aaa
```

那么它就会等待来自标准输入的输入，并将其视为要搜索字符串 "aaa "的文件。使用 Ctrl-D（表

示 "文件结束"）告诉 grep 命令 "文件"（标准输入）已经结束，这样它就会开始搜索。您可以将标准输入重定向到文件
~/lab1/ab.txt

```
$ grep aaa < ~/lab1/ab.txt
```

该命令将把 ~/lab1/ab.txt 文件当作键盘输入的文件处理。当然，您也可以使用

```
$ grep aaa ~/lab1/ab.txt
```

来直接查找 **ab.txt** 中的字符串 "aaa"，但重定向输入的功能对连锁命令非常有用，正如我们稍后将看到的管道功能。

## 输出重定向 (>)

输出重定向与输入重定向相反。在这种情况下，可以将标准输出（即显示屏）的输出重定向到文件。例如，**echo** 命令向显示屏输出一个字符串。我们可以重定向它，使其输出到一个文件，比如 output.txt，例如

```
$ echo hello > output.txt
```

该命令会将 hello 的输出写入名为 output.txt 的文件。

## 管道 (|)

我们还可以将 X 程序的输出重定向到 Y 程序的输入。这就是所谓的 "管道"；我们在 X 和 Y 之间建立了一个 "管道"：

```
$ ls -l | grep alice
```

这两条命令由管道 `|' 分隔。第一条命令将输出文件列表及其属性，管道将输出作为标准输入发送给 "grep "程序。

## 将输出添加到文件 (>>)

操作符 >> 会将命令的输出附加到文件中。例如

```
$ echo 'abc' >> test.txt
```

会将文本 "abc "添加到文件 "text.txt "的末尾。

**练习 4.** cracklib-check 是一个检查密码强度的程序。它从标准输入（键盘）接收输入。如果密码正确，它将输出密码，并在密码末尾显示 OK 消息。本练习使用 /home/alice/lab1 中的文件 "passwords.txt"。使用 cracklib-check 程序和输入/输出重定向运算符，检查存储在 passwords.txt 中的密码，并只输出正确的密码条目。

## 3.环境变量

环境变量是可以在 shell 环境中使用的值的占位符。这些变量可用作宏来定义简单的数据（如字符串）或（shell 脚本）函数。bash 环境有几个预定义的环境变量。下面我们将列出其中几个。

**主页（HOME）：**该环境变量包含当前用户的主目录。

**PATH：**该环境变量包含一系列目录，以冒号（':'）分隔。当用户在未指定可执行文件路径的情况下执行命令时，将查询该序列中的目录。bash 系统将在 PATH 环境变量中设置的目录中查找可执行文件。

**USER：**该环境变量包含当前登录用户的用户名。

要查看 shell 中当前定义的所有环境变量，请使用 "env "命令：

```
$ 环境
```

### 打印环境变量

通过在环境变量前添加"$"符号，可以访问该变量的值。例如，要显示 HOME 环境变量的内容，可以使用 echo 命令，如

```
$ echo $HOME
```

该命令将打印 HOME 环境变量的值。同样，命令

```
$ ls $HOME
```

将列出主目录的内容。

### 设置环境变量

您可以通过 "导出 "命令创建或更改环境变量的值。例如，执行以下命令可以将 PATH 设置为指向 /home/alice：

```
$ export PATH=/home/alice
```

**注意**这会覆盖 bash 环境的默认搜索路径，导致内置命令无法运行。更改 PATH 环境的更安全方法是在现有路径上添加额外路径：

```
$ export PATH=$PATH:/home/alice
```

PATH 中的路径以冒号": "分隔。

当用户在 bash shell 中输入命令时，bash 会根据 PATH 变量的内容，按从左到右的顺序搜索与命令相匹配的程序名，并执行第一个匹配的程序。

**练习 5.**将文件 ~/lab1/hello 重命名为 ~/lab1/ls，并改变 shell 的行为，使命令 "ls "指向 ~/lab1/ls，而不是 /usr/bin/ls。你使用什么命令来实现这一点？

## 编辑文件

到目前为止，我们还没有介绍如何在命令行界面（CLI）中直接编辑文件。CLI 中有几种常用的文本编辑器，例如 nano、vim 和 emacs。对于初学者，我们推荐使用 nano，因为与其他编辑器相比，它的用户界面相对友好。要编辑一个文件，比如 myfirstfile.txt，只需键入以下命令即可：

```
$ nano myfirstfile.txt
```

如果文件已经存在，它将打开它。否则，它将被视为一个新文件。界面如下


上图窗口底部显示了 nano 的几个主要命令。命令旁边的 ^ 符号，如 ^X 表示控制键。例如，要退出编辑器，只需同时按下控制键和 X 键。这些命令不言自明。我们将不再详细解释 nano 的功能。要在下面的练习中创建和编辑 shell 脚本，只要知道如何打开/写入/读取文件和退出文本编辑器就足够了。

# 4.外壳脚本

## 什么是 shell 脚本?

在交互模式下，用户一次输入一条命令，命令会立即执行并得到反馈，除此之外，Bash（与许多其他 shell 一样）还可以运行一整套命令脚本，即 "Bash shell 脚本"（或 "Bash 脚本 "或 "shell 脚本"，或简称 "脚本"）。shell 脚本可能只包含一个非常简单的命令列表，甚至只是一条命令，也可能包含函数、循环、条件结构。

在 Bash Shell 脚本中，第一行总是以 **#!/bin/bash** 开头。这就告诉系统，你的脚本是一个 bash 脚本。

## 小抄

有用的命令列表： https://devhints.io/bash

## 一些 shell 脚本示例

下面是一些 shell 脚本的示例。在本实验中，我们不会编写高级脚本。这里的目的是向学生介绍 shell 脚本的概念，以便于讨论此类脚本的安全限制。有关 shell 脚本的更全面介绍，请参阅本指南末尾的参考资料。

下面的示例脚本位于实验室虚拟机的 **/home/alice/lab1/shell** 目录中。以下示例假定您已将该目录设为当前目录，例如使用命令

```
$ cd /home/alice/lab1/shell
```

## 你好，世界！

下面是一个 hello-world 程序的示例（在 hello.sh 文件中），它将 "Hello, World "打印到标准输出。

***hello.sh***

```
#!/bin/bash
echo "世界你好！"
```

可能需要将其转换为可执行文件，使用命令

```
$ chmod a+x hello.sh
```

(我们将在有关 Unix 安全的讲座中详细介绍 chmod 命令）。然后运行

```
$ ./hello.sh
```

## 变量

bash 脚本中的变量无需明确声明，只需在变量前加上 $ 符号即可读取。要初始化变量，请使用 = 操作符。

例如

```
$ X=1
$ echo $X
```

会将变量 X 初始化为 1 并显示在标准输出上。

## 算术

算术运算必须用双括号括起来。

例如：以下脚本

```
#!/bin/bash

X=1
((X=X+1))
echo $X
```

将 X 初始化为 1，递增 1 并打印其内容（将是 "2"）。

## 引言

在 bash 中，单引号告诉 bash 环境将引号之间的所有内容按原样处理。特别是，$ 符号将被忽略，因此引号中引用的（环境）变量不会被展开。

例如

```
$ echo '当前用户是 $USER。
```

将准确输出：

当前用户为 $USER

双引号告诉 bash，引号之间的某些字符具有特殊含义。特别是 $ 符号，bash 会将其解释为符号后面的内容是一个变量，并将其扩展为实际值。

例如

```
$ echo "当前用户是 $USER"。
```

将输出

当前用户是 alice。

## 顺序构图

操作符 ; （分号）告诉 bash，由分号分隔的命令将按顺序（从左到右）执行。

例如

```
$ echo hello; echo world
```

将执行两条 echo 命令，一条打印 hello，另一条打印 world。

当在一行中编写多条命令时，例如直接从命令提示符运行这些命令时，该功能非常有用。

## 和左侧的方括号 [

bash 脚本中最令人困惑的一点是括号和小括号的使用。与大多数其他编程语言不同，大括号/小括号并不是用来组合表达式的。左方括号 [ 实际上指的是文件 /usr/bin/[（没错，文件名实际上就是左方括号"["）。["更传统的名称是 "test "命令（/usr/bin/test）。要查看 test 命令的完整解释，请运行 "man test"。

当命令中使用左括号 [ 时，它会检查其最后一个参数是否为右括号"]"。这样做的目的只是给人一种表达式分组的错觉。更准确地说，命令 /usr/bin/[ 的语法如下

**[ 表达 ]**

其中 EXPRESSION 是任何测试。常用的两种测试形式是

- 字符串比较
    - 等价测试：例如，$x = "hello"
    - 测试不等式：例如，$x != "hello"
- 整数比较
    - 等价：例如，$x -eq 10

- 小于：例如，$x -lt 10
- 小于或等于：例如，$x -le 10
- 大于： $x -gt 10
- 大于或等于： $x -ge 10

命令 [ 可以用测试命令代替，省略右括号 ]。例如，以下两条命令可以互换。

```
[ $x -lt 10 ]
```

**测试 $x -lt 10**

test/[ 命令在条件和循环中发挥着重要作用。

测试/[ 命令可以使用 && （逻辑 AND）、|| （逻辑 OR）组合使用。例如

```
[ $x -lt 10 ] && [ $x -gt 5 ]
```

检查变量 x 是否在 (5,10) 范围内。等同于使用 test 命令：

```
test $x -lt 10 && test $x -gt 5
```

## IF-THEN-ELSE

if-then-else 命令的形式是

```
if test-command
则
    指令
不然
   指令
fi
```

其中，test-command 是使用上一节讨论的 "test "或"[]"命令形成的任何命令。

如有必要，可以省略 "其他 "部分：

```
if test-command
则
    指挥部
fi
```

下面是 if-then-else 的示例（~/lab1/shell/check_num**.sh**）：

```
#!/bin/bash
echo "键入输入的整数，然后按 [Enter] 键：" 读 x
if [ $((x%2)) -eq 0 ]
      则 echo "$x 为偶数" else
      echo "$x 为奇数"
fi
```

**读 x** "命令从标准输入（如键盘）中读取输入，并将其存储到变量 x 中。

命令 **elif**（即 "else if"）可用于简洁地编写嵌套的 if-then-else 脚本。例如，以下两个脚本（testbinary.sh 和 testbinary2.sh）是等价的：

**testbinary.sh**

```
#!/bin/bash

read x

if [ $x -eq 0 ]
then
   echo zero
elif [ $x -eq 1 ]
then
   请回答
   echo 'not binary' （非二
   进制
fi
```

**testbinary2.sh**

```
#!/bin/bash

read x

if [ $x -eq 0 ]
then
   echo zero
else
   if [ $x -eq 1 ]
   then
      请回答
      echo 'not binary' （非
      二进制
  fi
fi
```

**练习 6.**编写一个 shell 脚本，将从标准输入中读取的数字分数转换成 ANU 分级：HD（80 - 100），D（70 - 79），CR（60 - 69），P（50 - 59），N（0 - 49）。

## FOR-LOOPS

**for 循环的语法**

**for** *loop-variable* **in** *value-range*
**做**
    *指挥部*
**完成的**

值范围可以明确枚举，如

```
for i in 1 2 3 4
do
   echo "iteration $i"
done
```

或指定为一个时间间隔：

```
for i in {1...4}
do
   echo "iteration $i"
done
```

或者，我们可以使用更熟悉的 C 语言或类似 Java 的语法：

```
for ((i=1; i<=4; ++i))
do
  echo "iteraton $i"
done
```

## WHILE-LOOPS

while 循环的语法

**当***测试命令*

**做**

*指挥部*

**完成的**

例如

```
i=1
while [ $i -le 4 ]
do
  echo "iteration $i"
  ((i++))
完成的
```

**练习 7.**编写一个 shell 脚本，打印斐波那契数列的前 n 个元素，其中 n 从标准输入读取。

## 更多 shell 脚本示例和练习（可选）

如果您觉得前面的练习太简单，这里有一些高级示例和练习供您尝试。这些都是可选的，在实验过程中不会涉及。本课程的所有考核项目都不依赖于这些示例，因此您可以放心地跳过这部分内容。

**loop_examples.sh**：

```
#!/bin/bash
#The following for loop will clear all files and folders
#under folder_to_be_cleared directory #下面的 for 循环将清除文件
夹_to_be_cleared 目录下的所有文件和文件夹

for i in $(ls
folder_too_be_cleared/) do
    rm -rf folder_too_be_cleared/$i
完成的

#syntax {Start...End...Increment}
for i in {1...5...1}
做
    echo "欢迎 $i 次"
完成的

#calculate 1+2+...+10
counter=1
总和=0
while [ $counter -le 10 ]
do
    sum=$((sum+counter))
   ((counter++))
完成的
echo "1+2+...+10=$sum"
```

该文件包含 3 个循环。第一个循环将找出 **folder_too_be_cleared** 目录下列出的所有文件和文件夹，并使用这些文件和文件夹构建 **rm** 命令，以删除目录下的所有文件和文件夹。第二个循环是一个简单的 for 循环，如果你以前学过 C/Java/Python 语言的话。最后一个是 while 循环，它将 1 到 10 的所有数字相加。

**read_nums.sh：**

```
#!/bin/bash
#usage：./read_nums.sh [输入文件]
Filename=$1
if [ -z $1 ]
;
        然后 echo "未找到输入"
不然
num=( $(<$Filename) )
numlen=${#num[*]}

echo "源文件中共有"$numlen "个数字。它们是：" echo ${num[*]}；
echo "而按相反顺序排列，它们是："

for ((i=1;i<=$numlen;i++))
        do
        echo -ne ${num[$numlen-$i]}" 已完
        成完成
echo
fi
```

执行：

```
$ ./read_nums.sh input_nums.txt
```

在本例中，input_nums.txt 包含 8 个数字。程序读取所有数字，并按相反顺序输出。

**练习 8 (*)。**在 ~/lab1/ 目录中有一个文件 **numbers.txt**，其中包含一个数字列表。编写一个冒泡/选择排序 shell 脚本，将数字按升序排序，并保存在 sorted.txt 中。

## 参考文献


- Stephen G. Kochan 和 Patrick Wood.Unix、Linux 和 OS X 中的 Shell 编程（第 4 版）。Addison-Wesley Professional, 2016.
- Machtelt Garrels."Linux 入门：实践指南》。2010. http://tille.garrels.be/training/tldp/

- Arnold Robins."Unix in a Nutshell》，第 4 版，O'Reilly 出版社，2005 年。
- Mike G. "BASH 编程：Introduction HOW-TO", http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html