

2 初识Shell脚本

手动运维的缺陷：效率低、风险高

一、Shell概述

1.1 什么是Shell?

现在人们通常使用的操作系统都带有图形界面，简单直观，容易上手。然而**早期**的**计算机并没有图形界面**，人们只能使用烦琐的**命令**来控制计算机。

真正能够控制计算机硬件（CPU、内存、硬盘）的只有**操作系统内核**

（Kernel），**图形界面**和**命令行**都是架设在**用户和内核之间的桥梁**，是为了方便用户控制计算机而存在的。

由于安全等原因，**用户不能直接接触内核**，因此需要在用户和内核之间增加“命令解释器”，这既能简化用户的操作，又能保障内核的安全。这个**命令解释器叫作“Shell”**，它能让用户更加高效、安全、低成本地使用内核。

💬 Shell（外壳）的概念与Kernel（内核）相对应！

💬 Windows中的PowerShell、CMD也算是“Shell”。

1.2 Shell如何连接用户和内核

Shell能够接收用户输入的命令，并对命令进行处理，处理完毕后再将结果反馈给用户，如输出到显示器、写入文件等。

Shell本身的功能很弱，**文件操作、输入输出、进程管理等都得依赖内核**。用户运行一个**命令**，大部分情况下Shell都会去**调用内核暴露出来的接口**，这就是在使用内核，只是这个过程被Shell隐藏了起来，在背后默默进行，用户看不到而已。

接口其实就是一个一个的函数，使用内核就是调用这些**函数**，除了函数没有别的途径使用内核。

⚠ 即通过**命令**调用**内核接口**实现具体操作。

例如，用户在Shell中输入 `cat log.txt` 命令就可以查看 `log.txt` 文件中的内容。`log.txt` 放在磁盘的哪个位置？分成了几个数据块？如何操作磁头读取它？这些底层细节Shell统统不知道，它只能去调用内核提供的 `open()` 和 `read()` 函数，告诉内核读取 `log.txt` 文件；然后内核按照Shell的指令去读取文件，并将读取到的文件内容交给Shell；最后由Shell把文件内容呈现给用户（呈现到显示器上还得依赖内核）。

1.3 Shell如何连接其他程序

在Shell中输入的命令，有一部分是**Shell本身自带的**，这叫作**内置命令**；有一部分是**其他应用程序**（**一个程序就是一个命令**），这叫作**外部命令**。

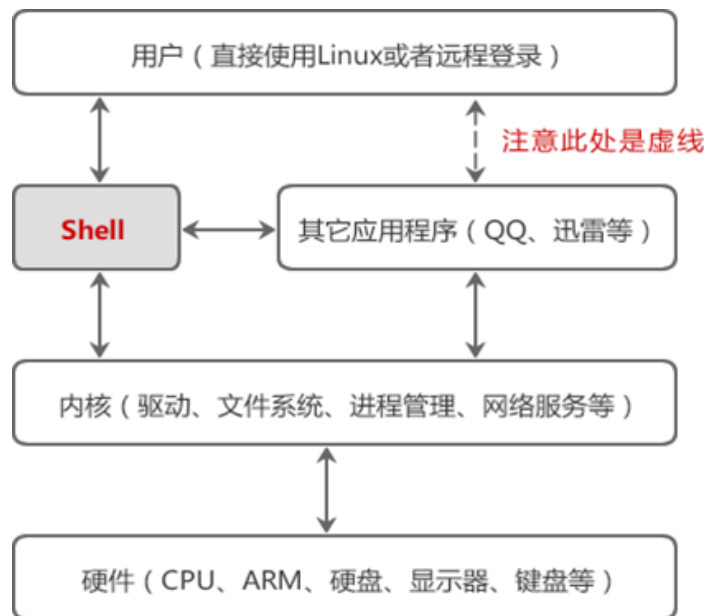
💬 通过 `type` 命令可以检查命令是否是内置命令。

💬 通过 `help` 命令可以查看shell内置命令列表及帮助。

Shell本身支持的命令并不多，功能也有限，但是Shell可以调用其他程序，每个程序就是一个命令，这使得Shell命令的数量可以无限扩展，其结果就是Shell的功能非常强大，完全能够胜任Linux的日常管理工作，包括文本或字符串检索、文件的查找或创建、大规模软件的自动部署、更改系统设置、监控服务器性能、发送报警邮件、抓取网页内容、压缩文件等。

💬 Shell还可以让多个外部程序发生连接，在它们之间很方便地传递数据，也就是把一个程序的输出结果传递给另一个程序作为输入信息。

Shell连接程序的示意图如下图所示。注意**用户**和**其他应用程序**是通过虚线连接的，因为用户启动Linux后直接面对的是Shell，通过Shell才能运行其他应用程序。



1.4 Shell也是一种脚本语言

用户不但可以在**Shell中输入命令**，还可以将**内置命令**和**外置命令**编写在**文件**中，然后**利用Shell进行运行**，这种方式与使用C++、C#、Java、Python等**常见的编程语言**并没有什么两样。

Shell通过**内置命令**支持以下基本的**编程元素**。

- 流程控制结构：if...else选择结构，case...in开关语句，for、while、until循环。
- 变量、数组、字符串、注释、加减乘除、逻辑运算等概念。
- 函数，包括用户自定义的函数和内置函数（如 printf()、export()、eval()等）。

💬 shell内置命令除少量基础命令外，绝大多数用于支持编程元素。

⚠️ 因此，也可以说Shell也是一种编程语言，它的解释器是Shell程序。

编程语言中有一部分语言，如C/C++、Pascal、Go、汇编语言等，都必须在程序运行之前将所有代码翻译成二进制形式，也就是生成可执行文件。用户拿到的是生成的可执行文件，看不到源码。这个过程叫作编译，这样的编程语言叫作**编译**

型语言，完成编译过程的软件叫作编译器。编译型语言的优点是执行速度快、对硬件要求低、保密性好，适合开发操作系统、大型应用程序等。

而有的编程语言，如Shell、JavaScript、Python等，需要一边执行一边翻译，不会生成可执行文件，用户必须拿到源码才能运行程序。程序开始运行后会即时翻译，翻译完一部分执行一部分，不用等到所有代码都翻译完。这个过程叫作解释，这样的编程语言叫作**解释型语言或者脚本语言**（Script），完成解释过程的软件叫作解释器。脚本语言的优点是使用灵活、部署容易、跨平台性好，非常适合Web开发以及小工具的制作。

⚠ Shell就是一种**脚本语言**，用户编写完源码后不用编译，直接运行源码即可。因此，利用Shell编写的程序也叫做**Shell脚本**。

1.5 Shell是运维工程师必备技能

Shell主要用来开发一些实用的、自动化的**小工具**，而不是用来开发具有复杂业务逻辑的中大型软件。

例如，检测计算机的硬件参数、搭建Web运行环境、日志分析等，Shell都非常合适。

使用Shell的熟练程度反映了用户对Linux的掌握程度，运维工程师、网络管理员、程序员都应该学习Shell。

对Linux运维工程师来说，Shell更是必须掌握的技能。Shell脚本很适合处理**纯文本**类型的数据，而Linux中绝大多数配置文件、日志文件（如NFS、rsync、HTTPD、Nginx、MySQL等）、启动文件都是纯文本类型的文件。

二、编写Shell脚本

2.1 Shell的各种版本

Linux支持的Shell的版本有很多种，其中常用的几种是Bourne Shell、C Shell和Bash Shell。

- Bourne Shell简称**sh**，由贝尔实验室开发，是**UNIX最初使用的Shell**，并且在每种UNIX上都可以使用。Bourne Shell在编程方面相当优秀，可以满足用

户大部分的Shell编程要求，也是平时工作中比较常用的Shell版本。

- C Shell简称**cs**h，比Bourne Shell更加适用于编程。C Shell的语法与**C语言**的语法很相似。Linux操作系统还为喜欢使用C Shell的人提供了**tc**sh。tcsh是C Shell的一个扩展版本，包含命令行编辑、可编程单词补全、拼写校正、历史命令替换、作业控制和类似C语言的语法。它不仅兼容Bash Shell提示符，还提供比Bash Shell更多的提示符参数。
- Bash Shell（Bourne Again Shell）是Linux的**默认Shell**，它是Bourne Shell的扩展，简称bash。bash与Bourne Shell完全向下兼容，也就是说bash可以兼容相同版本的Bourne Shell。bash在Bourne Shell的基础上增加、增强了很多特性。bash有许多特色，提供如命令补全、命令编辑和命令历史表等功能，而且还具备C Shell的很多优点，有灵活和强大的编程接口，同时又有很友好的用户界面。

⚠ **Bash Shell 是 Linux 的默认Shell，本课程编写的Shell脚本基于Bash。**

2.2 查看Shell版本

⚠ **Shell是一个程序，一般放在 `/bin` 或者 `/usr/bin` 目录下。**

当前Linux系统**可用的Shell**都记录在 `/etc/shells` 文件中，它是一个**纯文本文件**，可以使用 `cat` 命令查看它。

```
1 [root@Shell ~]# cat /etc/shells
2 /bin/sh
3 /bin/bash
4 /usr/bin/sh
5 /usr/bin/bash
```

💬 在现代Linux中，`sh` 已经被 `bash` 代替，`/bin/sh` 往往是指向 `/bin/bash` 的符号链接。

```

1 # which命令可查看命令对应程序所在路径
2 [root@Shell ~]# which sh
3 /usr/bin/sh
4 [root@Shell ~]# ll /usr/bin/sh
5 lrwxrwxrwx. 1 root root 4 Jan 2 19:00 /usr/bin/sh -> bash

```

💬 如果用户希望查看当前Linux的默认Shell，那么可以输出 `SHELL` 环境变量。

```

1 # 查看SHELL环境变量
2 [root@Shell ~]# echo $SHELL
3 /bin/bash
4 [root@Shell ~]# ll /bin
5 lrwxrwxrwx. 1 root root 7 Jan 2 19:00 /bin -> usr/bin

```

2.3 第一个Shell脚本

打开文本编辑器(可以使用 vi/vim 命令来创建文件)，新建一个文件 `test.sh`，扩展名为 `sh` (sh代表Shell)，但是扩展名并不影响脚本执行！

📝 待学习完运行脚本后，请验证更改扩展名是否影响脚本执行。

```

1 # 创建一个简易脚本
2 [root@Shell ~]# vi test.sh
3 #! /bin/bash
4 # first script
5 echo "abc "

```

- 第 1 行的 `#!` 是一个约定的标记 (shebang行)，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell；后面的 `/bin/bash` 就是指明了解释器的具体位置。

⚠️ shebang行 功能相当于Windows系统中的文件关联方式。

如果省略shebang行默认使用bash运行文件，但是建议在脚本中指定解释器。

- 第2行的内容为注释。Shell 脚本中所有以 `#` 开头的都是**注释**（以 `#!` 开头的除外）。
- 第3行的 `echo` 命令用于向标准输出文件输出文本。

⚠ Shell脚本通过**Shell内置命令**组织各类**外置命令**以完成复杂的任务。**命令式编程的特点所决定**。

💬 在 `.sh` 文件中使用命令与在终端直接输入命令的效果是一样的。

📖 课外拓展：了解命令式编程与函数式编程范式的差异。

三、运行Shell脚本

运行 Shell 脚本有两种方法。

- 在新进程中运行：脚本运行时会开启一个新bash进程。
- 在当前 Shell 进程中运行。

3.1 在新进程中运行

3.1.1 Shell 脚本作为可执行文件运行

⚠ Shell 脚本也是一种**可执行文件**，可以在直接调用（需要使用 `chmod` 命令给脚本**加上执行权限**）。

```
1  # 为test.sh增加执行权限
2  [root@Shell ~]# chmod +x ./test.sh
3  # 执行脚本
4  [root@Shell ~]# ./test.sh
5  abc
```

第1行中，`chmod +x` 表示给 `test.sh` 增加执行权限。

第2行中，`./` 表示当前目录，整条命令的意思是执行当前目录下的 `test.sh` 脚本。如果不写 `./`，Linux 会到系统路径（由 `PATH` 环境变量指定）下查找 `test.sh`，而系统路径下显然不存在这个脚本，所以会执行失败。

⚠ 通过这种方式运行脚本，**脚本文件第一行的 `#!/bin/bash` 一定要写对**，**好让系统查找到正确的解释器。**

案例：shebang行错误的结果

脚本内容被当做Python程序运行！

```
Shell | 复制代码
1 [root@Shell ~]# vi test.sh
2  #! /usr/bin/python
3  echo "abc"
4 [root@Shell ~]# ./test.sh
5  File "./test.sh", line 2
6      echo "abc"
7      ^
8  SyntaxError: invalid syntax
```

3.1.2 将 Shell 脚本作为参数传递给 Bash 解释器

⚠ 也可以**直接运行** Bash 解释器，将**脚本文件的名称**作为**参数**传递给 Bash。

```
Shell | 复制代码
1 [root@Shell ~]# /bin/bash test.sh
2  abc
```

⚠ 这种脚本运行方式，**不需要在脚本文件的第一行指定解释器信息**，**写了也没用。**

这两种写法在本质上是相同的。

- 第一种写法给出了绝对路径，会直接运行 Bash 解释器。
- 第二种写法通过 bash 命令找到 Bash 解释器所在的目录，然后再运行，只不过多了一个查找的过程而已。


```

1 [root@shell ~]# ps aux | grep bash
2 root      1398  0.0  0.0 115544  2096 pts/0    Ss   Jan01   0:00 -bash
3 root      1656  0.0  0.0 115544  1988 pts/1    Ss+  Jan01   0:00 -bash
4 root      4430  0.0  0.0 115544  1980 pts/2    Ss   23:48   0:00 -bash
5 root      4447  0.0  0.0 112808   964 pts/2    S+   23:51   0:00 grep --c
    olor=auto bash
6 [root@shell ~]# bash
7 [root@shell ~]# ps aux|grep bash
8 root      1398  0.0  0.0 115544  2096 pts/0    Ss   Jan01   0:00 -bash
9 root      1656  0.0  0.0 115544  1988 pts/1    Ss+  Jan01   0:00 -bash
10 root     4430  0.0  0.0 115544  2024 pts/2    Ss   23:48   0:00 -bash
11 root     4448  0.0  0.0 115544  1996 pts/2    S    23:51   0:00 bash
12 root     4458  0.0  0.0 112808   964 pts/2    S+   23:51   0:00 grep --c
    olor=auto bash
13 [root@shell ~]# exit
14 exit
15 [root@shell ~]# ps aux | grep bash
16 root      1398  0.0  0.0 115544  2096 pts/0    Ss   Jan01   0:00 -bash
17 root      1656  0.0  0.0 115544  1988 pts/1    Ss+  Jan01   0:00 -bash
18 root      4430  0.0  0.0 115544  2028 pts/2    Ss   Jan07   0:00 -bash
19 root      4475  0.0  0.0 112808   964 pts/2    S+   00:04   0:00 grep --c
    olor=auto bash

```

3.2 在当前进程中运行


⚠ `source` 是 Shell 内置命令，它会读取脚本文件中的代码，并依次执行所有语句。

`source` 命令会强制执行脚本文件中的全部命令，而忽略脚本文件的权限。

`source` 命令的语法格式为：`source filename` 或 `. filename`。


⚠ 对于第二种写法，注意点号 `.` 和文件名中间有一个空格。

```
1 [root@Shell ~]# ll test.sh
2 -rwxr-xr-x. 1 root root 30 Jan  3 01:38 test.sh
3 [root@Shell ~]# chmod -x test.sh
4 [root@Shell ~]# ll test.sh
5 -rw-r--r--. 1 root root 30 Jan  3 01:38 test.sh
6 [root@Shell ~]# ./test.sh
7 -bash: ./test.sh: Permission denied
8 [root@Shell ~]# source ./test.sh
9 abc
10 [root@Shell ~]# . ./test.sh
11 abc
12 [root@Shell ~]# . test.sh
13 abc
```

 课外练习：检测上述运行方式是否启动了新进程。

四、VI编辑器（补充）

Linux首选的文本编辑器是Vim，它是一个基于文本界面的编辑工具，使用简单且功能强大，更重要的是，大多数 Linux 的发行版配备的都是 Vim。

 本课程优先使用Vim编写**代码量较小**的Shell脚本。

4.1 VI的三种模式

Vim 有 3 种工作模式：命令模式（或称常规模式）、插入模式、编辑模式（或称末行模式）。

- **命令模式**：Vim 启动后，默认进入命令模式，在任何模式下，都可以按 **Esc** 键返回到命令模式，可以多按几次 **Esc** 键，保证顺利返回到命令模式。在此模式下，可以使用上、下、左、右键或者 **k**、**j**、**h**、**l** 键进行光标移动，也可以键入不同的命令完成选择、复制、粘贴、删除等操作。
- **插入模式**：在插入模式下可以编辑文本内容。在命令模式下按 **i**、**a**、**o** 等键可以进入插入模式，在此模式下可以输入文本，但命令执行后的字符插入位置不同。
- **编辑模式**：在命令模式下按 **:** 键进入编辑模式。这时光标会移到屏幕底部，

在这里可以输入相关指令保存修改或退出 Vim，也可以设置编辑环境、寻找字符串、列出行号等。指令执行后会自动返回命令模式。

三种模式的可以通过vim底部区分！

- 命令模式
- 插入模式
- 编辑模式

4.2 VI的基本操作

4.2.1 打开/新建文件

打开文件的命令格式：`vi 文件路径`

- 当文件路径存在时，则在vim中读取文件内容
- 当文件路径不存在时，则在对应路径新建文件

刚打开文件时进入的是命令模式，此时文件的下方会显示文件的一些信息，包括文件名、文件的总行数和字符数，以及当前光标所在的位置等。

4.2.2 编辑文件

vim打开后默认进入命令模式，无法直接输入内容。

从命令模式进入输入模式进行编辑，可以按下 `I`、`i`、`O`、`o`、`A`、`a` 等键来完成，不同的键只是光标所处的位置不同而已。

- `i` 为从目前光标所在处输入
- `I` 为在目前所在行的第一个非空格符处开始输入
- `a` 为从目前光标所在的下一个字符处开始输入
- `A` 为从光标所在行的最后一个字符处开始输入
- `o` 为在目前光标所在的下一行处输入新的一行
- `O` 为在目前光标所在的上一行处输入新的一行

当进入输入模式后，在Vim编辑窗口的左下角会出现“INSERT”标志。此时就可以对文件进行编辑。

4.2.3 保存/退出文件

文件编辑完成后，我们需要保存和退出文件。

Vim的保存和退出是在编辑模式中进行的，为了方便记忆，只需要记住 `w`、`q`、`!` 三个符号的含义即可完成保存任务。

- `w` 保存
- `q` 退出
- `!` 强制操作

这些命令会一些常用的组合：

- `q` 当文件没有修改时，直接退出
- `wq` 保存文件后退出
- `q!` 当文件被修改时，强制不保存退出
- `wq!` 强制保存后退出

4.2.4 其他常用命令

- `gg` 移动到文件的第一行
- `G` 移动到文件的最后一行
- `+` 光标移动到非空格符的下一行
- `-` 光标移动到非空格符的上一行
- `0` 或功能键 `Home` 移动到这一行的最前面字符处
- `$` 或功能键 `End` 移动到这一行的最后面字符处
- `nG` `n` 为数字。移动到这个档案的第 `n` 行
- `n` `n` 为数字。光标向下移动 `n` 行
- `dd` 删除光标所在的那一整行
- `ndd` `n` 为数字。删除光标所在的向下 `n` 行，例如 `20dd` 则是删除 `20` 行
- `yy` 复制光标所在的那一行
- `p` 将已复制的数据贴在光标下一行
- `P` 将已复制的数据贴在光标上一行
- `u` 复原前一个动作
- `[Ctrl]+r` 重做上一个动作
- `:set nu` 显示行号
- `:set nonu` 与 `set nu` 相反，为取消行号！

小结

- Shell概述： 内置命令， 外置命令， 脚本语言
- 编写Shell脚本： Shell版本， Shell脚本基本格式
- 运行Shell脚本： 各种运行方式的差异
- VI编辑器： 模式， 常用命令

课程目标

- 知识目标： 了解shell脚本的概念， 掌握shell脚本的编写方法和运行方法。
- 技能目标： 能够编写简单的shell脚本并运行。

课外拓展

- 进一步了解shell脚本的基本结构
- Bash快捷键

参考资料

- bash帮助： `man bash`