

《Linux从零入门实战》 L4

Shell编程快速入门

刘贵学

Shell编程课程大纲

☑ Part1 : Shell 简介

- ▶ 什么是Shell
- ▶ 为什么学Shell
- ▶ 如何学习Shell
- ▶ Hello World

☑ Part2: Shell 基础语法

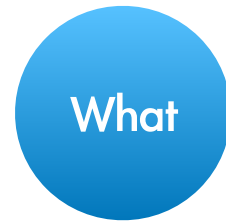
- ▶ 脚本建立与执行
- ▶ 变量
- ▶ 条件测试
- ▶ 循环语句
- ▶ 函数
- ▶ 常用库

☑ Part3: 实例场景

- ▶ Hello World
- ▶ 猜数字
- ▶ 文件读写
- ▶ 网络侦测
- ▶ 综合实例

Part1

Shell 介绍



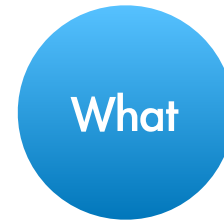
Shell 是什么？



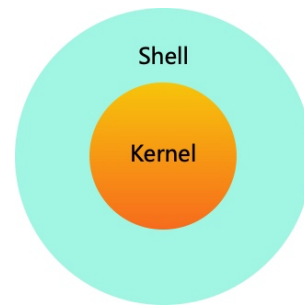
为什么要学 Shell？



如何学好 Shell？



Shell 是什么？



Linux 系统就像花生核桃一样，有核有壳

内核，Kernel，Linus 他们只聚焦内核开发

外壳，Shell，有命令行，界面的shell

Shell 指一种应用程序，它提供了一个界面

用户通过这个界面访问操作**系统内核**的服务

Shell 脚本种类

Shell 脚本 (shell script)

用shell 编写的脚本程序。
shell 通常都是指 shell 脚本

```
guixue@linux:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
/usr/bin/tmux
```

```
guixue@linux:~$ echo $SHELL
/bin/bash
```



为什么要学Shell？

Linux的精髓是什么？

将多个程序(命令)组装成大型程序, 而Shell 就是最好粘合剂

《Unix编程艺术》之 多道程序设计

Shell的优点

简单、高效、易维护、随写随用

Linux 越用越简单

自动化、可定制 -> 提高效率 -> 更简洁高效地使用Linux



如何学好Shell？



1. Shell 本身如果细讲，就可以做一个系列课程，我们只有 2 小时；
2. 2-8，用 20% 的时间，让大家熟悉 shell 80% 最常用的功能 即为重点，难点部分只介绍不细扣，留下自学。
3. 以例子驱动，根据应用场景，将知识点串起来，例子 场景介绍，问题建模分析，伪码逻辑，代码实现，运行效果，收获总结讲评。

脚本实例场景驱动



需求分析



伪码流程逻辑



总结



问题建模



代码实现与运行

编码规范

☑ 多加注释说明

☑ 命名建议规则

▶ 变量名大写

▶ 局部变量小写

▶ 函数名小写

▶ 函数变量 加 local

编辑器 VS Code

☒ 宿主机中写的代码，同步到虚拟机中

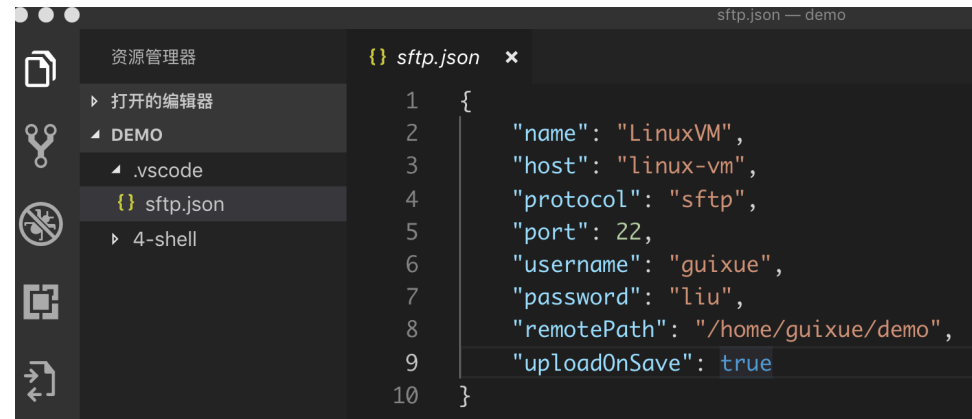
▶ 共享文件夹

▶ 配置 sftp



sftp 配置

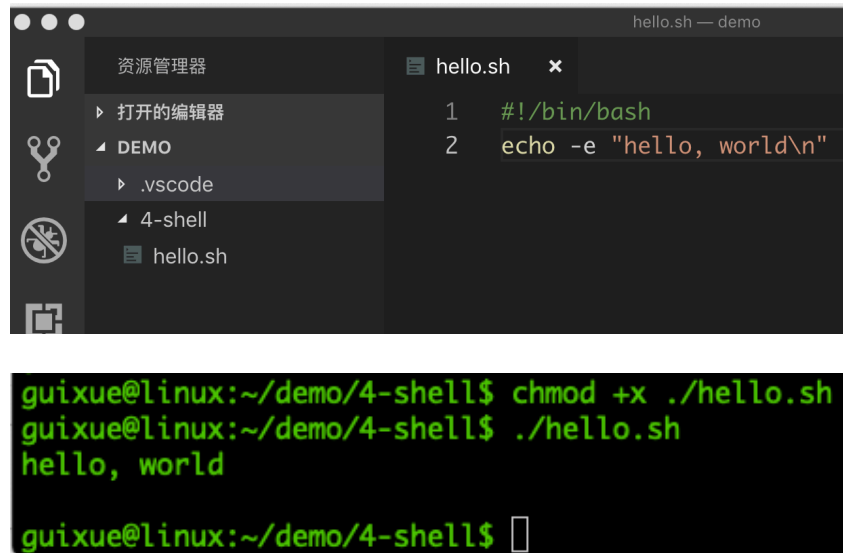
✓ 查看 -> 命令面板 -> SFTP config



The screenshot shows the Visual Studio Code editor interface. On the left, the 'Resource Manager' (资源管理器) sidebar is open, showing a project structure with a 'DEMO' folder containing '.vscode', 'sftp.json', and '4-shell'. The 'sftp.json' file is selected. The main editor area displays the content of 'sftp.json' with the following JSON configuration:

```
1  {
2      "name": "LinuxVM",
3      "host": "linux-vm",
4      "protocol": "sftp",
5      "port": 22,
6      "username": "guixue",
7      "password": "liu",
8      "remotePath": "/home/guixue/demo",
9      "uploadOnSave": true
10 }
```


Hello World



```
hello.sh — demo
资源管理器
  打开的编辑器
    DEMO
      .vscode
      4-shell
        hello.sh
hello.sh  x
1  #!/bin/bash
2  echo -e "hello, world\\n"

guixue@linux:~/demo/4-shell$ chmod +x ./hello.sh
guixue@linux:~/demo/4-shell$ ./hello.sh
hello, world

guixue@linux:~/demo/4-shell$
```

按惯例，每个语言都用 Hello，World 例子开始，为什么？

一件事情有好多层意思，仁者见仁，没有

直接意思，可以就是翻译过来，很简单：你好，世界。

引申出的意思是 一切正常，可以开始了。

背景含义是： 维基百科上说法，最早出现于B语言的示例，因为C语言之父就这样写的，沿用前人的做法，向前人致敬。

就像来到一个婴儿世间哇哇哭叫一样，不会说话，不会走路，第一次发声，此刻很有纪念意义。因为人生之路由此开启，以为也必然经历 就像《流浪地球》里的希望一样。

我们学习和使用一门技术之前，用它跟世界先问声好，立一个flag，咱们学习之路将由此刻正式开启。

注释

☒ #单行注释

▶ 行首解释说明

☒ 多行注释 (小技巧)

```
1- comment.sh x
1  #!/bin/bash
2
3  # 单行注释
4
5
6  :<<EOF
7  多行注释
8  注释内容...
9  注释内容...
10 注释内容...
11 EOF
12
```

变量

☑ 可以 改变 的量，类似数学里的 x , y

▶ `x=1`

☑ 命名只能使用英文字母，数字和下划线

☑ 首个字符不能以数字开头

☑ 中间不能有空格，可以使用下划线 (`_`)

☑ 不能使用bash里的关键字(help命令)

变量分类

局部
变量

环境
变量

特殊
变量

局部 变量

- ☑ 仅在当前shell实例中有效
- ☑ 在脚本或命令中定义
- ☑ 函数内的变量 加 local

变量：定义与使用

```
1  #!/bin/bash
2
3  # 1. 变量的定义 和使用
4  my_name=qiye
5  echo 1. ${my_name}
6
7  # 2. 另一种定义方式
8  course="linux start"
9  echo 2. ${course}
```

```
11  # 3. 只读变量
12  readonly course
13  course="linux kernel"
14  echo 3. ${course}
15
16  # 4. 删除变量
17  unset course
18  echo 4. ${course}
```

```
guixue@linux:~/demo/L4-shell/ch2-variable$ ./2-variable.sh
1. qiye
2. linux start
./2-variable.sh: line 13: course: readonly variable
3. linux start
./2-variable.sh: line 17: unset: course: cannot unset: readonly variable
4. linux start
```

函数变量 local 的使用，我们再函数章节中给出。

环境 变量

- ☑️ 又叫全局变量
- ☑️ 所有的程序，都能访问环境变量
- ☑️ 建议全部大写
- ☑️ env
- ☑️ export
- ☑️ source 与 .

如果想在当前脚本执行脚本, 而不想启动一个新的 shell, 可以使用 source

```
3-env.sh ×  
1  #!/bin/bash  
2  
3  # 1. export  导出一个环境变量  
4  export MY_NAME="guixue"  
5  
6  # 2. 查找自定义的环境变量  
7  env | grep MY_NAME
```


特殊
变量


| 变量 | 含义 |
|------|---|
| \$0 | 当前脚本的文件名 |
| \$n | 传递给脚本或函数的参数。n 是一个数字，表示第几个参数。例如，第一个参数是\$1，第二个参数是\$2。 |
| \$# | 传递给脚本或函数的参数个数。 |
| \$* | 传递给脚本或函数的所有参数。 |
| \$@ | 传递给脚本或函数的所有参数。被双引号(" ")包含时，与 \$* 稍有不同，下面将会讲到。 |
| \$? | 上个命令的退出状态，或函数的返回值。 |
| \$\$ | 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。 |

```
1  #!/bin/bash
2  # 特殊变量 $0 表示 脚本文件的名称
3  echo "文件名称: $0"
4  # 特殊变量 $0 表示 脚本文件的名称
5  echo "参数1: $1"
6  echo "参数2: $2"
7  echo "全部参数: $@"
8  echo "参数个数 : $#"
```

```
guixue@linux:~/demo/L4-shell/ch2-variable$ ./4-special.sh aa bb cc
文件名称: ./4-special.sh
参数1: aa
参数2: bb
全部参数: aa bb cc
参数个数 : 3
```

基本运算

☒ 算术运算

 `expr` 求值操作

☒ 关系运算

☒ 布尔运算

☒ 字符串运算

☒ 文件测试运算

反斜线 (\) 是bash的转义字符，也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报，响铃

\b 退格（删除键）

\f 换页(FF)，将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符（tab键）

\v 垂直制表符

算术运算

☑ + 加

☑ - 减

☑ * 乘

☑ / 除

☑ % 取余

```
1  #!/bin/bash
2  # 算术基本运算，加减乘除取余
3  a=11
4  b=5
5
6  # 加法 expr
7  val=`expr $a + $b`
8  echo "$a + $b = $val"
9
10 # 另一种数值运算 ${var}
11 val=$((a-b))
12 echo "$a - $b = $val"
```

反斜线 (\) 是bash的转义字符，也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报，响铃

\b 退格（删除键）

\f 换页(FF)，将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符（tab键）

\v 垂直制表符

关系运算

☑ -eq 相等

☑ -ne 不相等

☑ -gt 大于

☑ -ge 大于等于

☑ -lt 小于

☑ -le 小于等于

```
1  #!/bin/bash
2  # 关系运算
3  a=11
4  b=5
5
6  #例子, 其他 ne, gt, ge, lt, le 类似
7  if [ $a -eq $b ]
8  then
9      echo "$a -eq $b : a 等于 b"
10 else
11     echo "$a -eq $b: a 不等于 b"
12 fi
```

反斜线 (\) 是bash的转义字符, 也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报, 响铃

\b 退格 (删除键)

\f 换页(FF), 将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符 (tab键)

\v 垂直制表符

布尔与逻辑运算

☑ ! 非运算 [! false]

☑ -o 或运算

☑ -a 与运算 [\$a -lt 100 -a \$b -gt 15]

☑ && 逻辑与 [[\$a -lt 100 && \$b -gt 100]]

☑ || 逻辑或

☑ == 相等(数字)

☑ != 不等(数字)

```
1  #!/bin/bash
2  # 布尔运算
3  a=11; b=5
4  #例子, 其他 ne, gt, ge, lt, le 类似
5  if [ $a -eq $b ]
6  then
7      echo "$a -eq $b : a 等于 b"
8  else
9      echo "$a -eq $b: a 不等于 b"
10 fi
11
12 # 逻辑运算 注意 两个 [[ ]]
13 if [[ $a -gt 0 && $b -gt 0 ]]
14 then
15     echo "a, b 都大于 0"
16 fi
```

反斜线 (\) 是bash的转义字符, 也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报, 响铃

\b 退格 (删除键)

\f 换页(FF), 将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符 (tab键)

\v 垂直制表符

文件测试运算

☑️ -d 是否为目录

☑️ -f 是否为普通文件

☑️ -r, -w, -x 是否可读

▶️ -w, -x 可写, 可执行

☑️ -s 文件是否为空

☑️ -e 文件是否存在

```
1  #!/bin/bash
2  # 文件测试运算
3  file=$0 #
4
5  echo "文件是:$file"
6  if [ -f $file ]
7  then
8      echo "为普通文件"
9  fi
10 if [ -e $file ]
11 then
12     echo "存在";
13 fi
14 if [ -r $file ]
15 then
16     echo "可读";
17 fi
```

反斜线 (\) 是bash的转义字符, 也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报, 响铃

\b 退格 (删除键)

\f 换页(FF), 将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符 (tab键)

\v 垂直制表符

字符串

☑ 单引号

▶ 原样输出，变量无效

☑ 双引号

▶ 可以包含变量

反斜线 (\) 是bash的转义字符，也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报，响铃

\b 退格（删除键）

\f 换页(FF)，将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符（tab键）

\v 垂直制表符


```
1-string.sh x
1  #!/bin/bash
2  course="Linux从零入门实战"
3
4  # 单引号
5  question='Linux 如何入门? $course'
6  echo $question
7
8  # 双引号
9  answer="请学习《$course》课程!"
10 echo $answer
11
12 # 字符串拼接
13 echo -e "拼接后一起输出:\n"$question"\n"$answer
```

```
guixue@linux:~/demo/L4-shell/ch3-string$ ./1-string.sh
Linux 如何入门? $course
请学习《Linux从零入门实战》课程！
拼接后一起输出：
Linux 如何入门? $course
请学习《Linux从零入门实战》课程！
```

```
2-adv.sh x
1  #!/bin/bash
2  # 字符串长度
3  str="helloworld"
4  echo "字符串$str"的长度为:"${#str}
5
6  # 获取子串, 从第1个字符开始, 截取3个。
7  echo "字符串$str"子串:"${str:1:3}
8
9  # 查找子串
10 matched=`expr index "$str" wo` # 输出 4
11 echo "字符串$str"查找wo的位置在"$matched
```

```
guixue@linux:~/demo/L4-shell/ch3-string$ ./2-adv.sh
字符串 helloworld 的长度为 :10
字符串 helloworld 子串 :ell
字符串 helloworld 查找 wo 的位置在 5
```

字符串运算符

☑ = 字符串是否相等 [\$a = \$b]

☑ != 字符串是否不相等 [\$a != \$b]

☑ -z 字符串长度是否为0 [-z \$a]

☑ -n 字符串长度是否不为0 [-n "\$a"]

☑ \$ 字符串是否为空 [\$a]

反斜线 (\) 是bash的转义字符，也叫逃逸字符或者转义字符。

\\ 反斜杠

\a 警报，响铃

\b 退格（删除键）

\f 换页(FF)，将当前位置移到下页开头

\n 换行

\r 回车

\t 水平制表符（tab键）

\v 垂直制表符

数组

☒ 定义, 下标从 0 开始

☒ 设置/读取 长度

☒ 读取数组所有元素 @

☒ 读取数组长度

```
1-define.sh x
1  #!/bin/bash
2  # 1. 数组的定义
3  arr=(aa bb cc "hello world")
4  # 2. 设置 元素
5  arr[2]="222"
6  # 3. 读取 元素
7  echo "下标为2的元素: "${arr[2]}
8  # 4. 读取 所有元素
9  echo "所有元素: "${arr[@]}
10 # 5. 获取数组长度
11 len=${#arr[@]}
12 echo "数组长度: $len"
```

```
下标为2的元素 : 222
所有元素 : aa bb 222 hello world
数组长度 : 4
```

分支

☒ if else

☒ case

```
1  #!/bin/bash
2
3  age=20
4  if [ $age -le 10 ] # <=10
5  then
6      echo "少年"
7  elif [ $age -le 20 ] # <=20
8  then
9      echo "青年"
10 elif [ $age -le 50 ] # <=50
11 then
12     echo "中年"
13 else # >50
14     echo "老年"
15 fi
```

```
2-case.sh x
1  #!/bin/bash
2
3  status=1
4  case $status in
5      0) echo "todo"
6          ;;
7      1) echo "doing"
8          ;;
9      2) echo "done"
10         ;;
11 esac
```

循环

- ☑ for ... in ... do .. done
- ☑ while ... do ... done
- ☑ until ... do ... done
- ☑ break

```
1  #!/bin/bash
2  arr=(aa bb cc)
3  for item in ${arr[@]}
4  do
5      echo "$item"
6  done
```

for 有很多种写法

<https://blog.csdn.net/babyfish13/article/details/52981110>

函数

☒ function 关键字可选

☒ 函数后面的 () 可选

```
1  #!/bin/bash
2  #函数的定义和调用
3  function myfun()
4  {
5      echo "这是 shell 函数!"
6  }
7
8  myfun
```



```
1  #!/bin/bash
2  #函数传参 和返回值
3  function add()
4  {
5      local ret=$(( $1+$2 ))
6      return $ret
7  }
8
9  add 5 8
10 echo $?
```

常用命令

☑️ printf

☑️ date

☑️ cut 截取 下标从1开始

▶️ echo guixue|cut -c2-3

☑️ 根据 date 获取 random

☑️ 文本处理三剑客

▶️ grep 查找

▶️ sed 编辑

▶️ awk 报告

Part3

实例场景

猜数字小游戏

1. 程序会随机从0-99产生一个数字，作为谜底
2. 用户输入可能数字
3. 程序判断，如果错误告知，是大还是小，用户可以根据提示继续猜；如果正确，告知猜测的次数，并退出程序。

获取当前CPU的使用率

- ☑ `cat /proc/stat | grep -B1 -m1 "cpu"`
- ☑ `cpuTotal=user+nice+system+idle+iowait+irq+softirq+steal_time+guest`
- ☑ `cpuUsed=user+nice+system+irq+softirq+steal_time+guest`
- ☑ `cpuUsed * 100 / cpuTotal`

探测本地网络

- ☒ ping 命令简介
- ☒ 探测某个ip地址是否通畅
- ☒ 侦测本网段内所有主机情况

写日志

☑ 日志文件： `./tmp/my.log`

☑ 存在与权限检查

▶ 判断 `tmp` 目录是否存在，不存在则新建；

▶ 判断 `my.log` 文件是否存在，不存在则新建；

▶ 判断 是否具备 读写权限

☑ 备份检查

▶ 如果日志文件大小超过 1K，自动备份

☑ 在文件尾追加信息，带上日志时间

总结

- ☒ 简介
- ☒ 基础语法
- ☒ 场景实例
- ☒ 课程作业

课程作业：服务器监控脚本

1. 获取CPU的使用率，超过设定值，则报警；
2. 获取内存使用率，超过设定值，则报警；
3. 获取磁盘容量使用率，超过设定值，则报警；
4. 检查网站存活状态，异常则报警；
5. 查看某进程(比如sshd)是否存在，异常则报警；
6. 将以上结果写入日志；

参考资料：<https://blog.csdn.net/hualusiyu/article/details/8555003>

参考资料

☒ [UNIX shell范例精解](#)

☒ [Runoob Shell 教程](#)

☒ [BASH Programming](#)