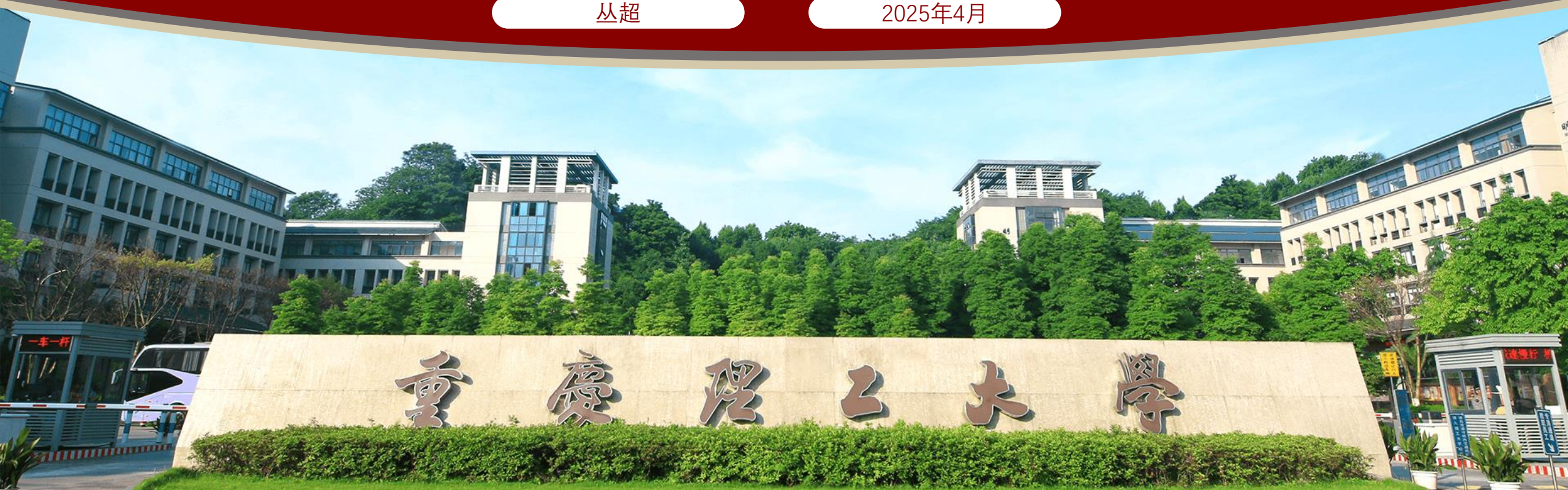
重庆理工大学/电气学院
CHONGQING UNIVERSITY OF TECHNOLOGY

嵌入式Linux系统开发教程
—基于ARM处理器通用平台（arm9-arm11-contexA系列）

丛超

2025年4月



1 常用Shell命令

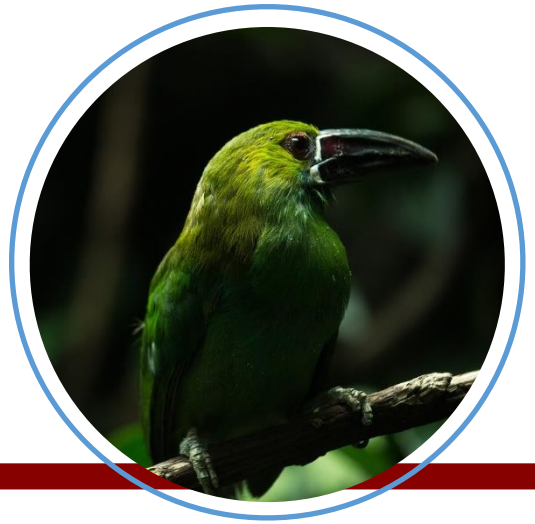
2 脚本编写基础

3 流程控制

01

常用Shell命令

- 对于任何想适当精通一些系统管理知识的人来说,掌握 shell 脚本知识都是最基本的,即使这些人可能并不打算真正的编写一些脚本。
- 想一下 Linux 机器的启动过程,在这个过程中,必将运行 /etc/rc.d 目录下的脚本来存储系统配置和建立服务。
- 详细的理解这些启动脚本对于分析系统的行为是非常重要的,并且有时候可能必须修改它。



- 学习如何编写 shell 脚本并不是一件很困难的事,因为脚本可以分为很小的块,并且相对于 shell特性的操作,只需要学习很小的一部分就可以了。
- 脚本的语法是简单并且直观的,编写脚本很像是在命令行上把一些相关命令和工具连接起来,并且只有很少的一部分规则需要学习。
- 绝大部分脚本第一次就可以正常的工作,而且即使调试一个长一些的脚本也是很直观的。

什么时候不使用 Shell 脚本?

01 大任务的数学操作

需要处理大任务的数学操作,尤其是浮点运算,精确运算,或者复杂的算术运算,(这种情况一般使用 C++或 FORTRAN 来处理)

02 资源密集型的任务

需要考虑效率时(比如,排序,hash 等等)

03 复杂的应用

- 必须使用结构化编程
- 多维数组
- 数据结构
- 产生或操作图形化界面 GUI

04 有跨平台移植需求

一般使用 C 或 Java

05 特殊任务

- 对于安全有很高要求的任务,比如你需要一个健壮的系统来防止入侵,破解,恶意破坏等等.
- 对于影响系统全局性的关键任务应用。
- 项目由连串的依赖的各个部分组成。

06 其它

- 直接操作系统硬件
- 需要 I/O 或 socket 接口
- 使用库或者遗留下来的老代码的接口
- 私人的,闭源的应用

Shell 脚本介绍：Bash

01

- Bash 是“Bourne-Again shell”首字母的缩写
- 也是 Stephen Bourne 的经典的 Bourne shell 的一个双关语。

02

- Bash 已经成为了所有 UNIX 中 shell 脚本的事实上的标准
- 其他一些 shell ，比如 ksh，zsh等

在shell脚本中可以使用任意的unix命令，有一些相对常用的命令。用来进行文件和文字操作。常用命令语法及功能如表3-1所示。

表 3-1 常用 shell 命令

函数	说明
echo "some text"	将文字内容打印在屏幕上
ls	文件列表
wc –l file	计算文件行数
wc –w file	计算文件中的单词数
wc -c file	计算文件中的字符数
cp sourcefile destfile	文件拷贝
mv oldname newname	重命名文件或移动文件
rm file	删除文件
grep 'pattern' file	在文件内搜索字符串比如：grep 'searchstring' file.txt
cat file.txt	输出文件内容到标准输出设备(屏幕)上
file somefile	得到文件类型
read var	提示用户输入，并将输入赋值给变量
sort file.txt	对 file.txt 文件中的行进行排序
uniq	删除文本文件中出现的行列比如：sort file.txt | uniq
expr	进行数学运算如: expr 2 "+" 3
find	搜索文件比如：根据文件名搜索 find . –name filename –print
tee	将数据输出到标准输出设备(屏幕) 和文件比如：somecommand | tee outfile
basename file	返回不包含路径的文件名比如：basename /bin/tux 将返回 tux
dirname file	返回文件所在路径比如：dirname /bin/tux 将返回/bin
head file	打印文本文件开头几行
tail file	打印文本文件末尾几行
sed	sed 是一个基本的查找替换程序。可以从标准输入(比如命令管道)读入文本，并将结果输出到标准输出(屏幕)。不要和 shell 中的通配符相混淆。比如：将 linuxfocus 替换为 LinuxFocus：cat text.file | sed 's/linuxfocus/LinuxFocus/' > newtext.file。
awk	awk 用来从文本文件中提取字段。缺省地，字段分割符是空格，可以使用-F 指定其他分割符。cat file.txt | awk -F, '{print ", " }'这里我们使用“, ”作为字段分割符，同时打印第一个和第三个字段。

02

脚本编写基础

特殊字符

脚本文件涉及特殊字符较多，本章重点介绍在脚本中出现频率较高的字符。

- **#** — 注释。以一个#开头的行(#!是例外)是注释行 — 注释也可以出现在一个命令语句的后面，注释行前面也可以有空白字符。
- **;** — 命令分割符[分号].
- **;;** — case语句分支的结束符[双分号]。
- **.** — “点”命令(圆点)。文件名的组成部分。 — 当点(.)以一个文件名为前缀时，起作用使该文件变成了隐藏文件。这种隐藏文件ls一般是不会显示出来的。作为目录名时，单个点(.)表示当前目录，两个点(..)表示上一级目录。

特殊字符

脚本文件涉及特殊字符较多，本章重点介绍在脚本中出现频率较高的字符。

- " — 部分引用(双引号)。 — "STRING"的引用会使STRING里的特殊字符能够被解释。
- ' — 完全引用(单引号)
- , — 逗号操作符(逗号)。逗号操作符用于连接一连串的数学表达式。
- \ — 转义符(后斜杠)。用于单个字符的引用机制。

变量和参数

1.变量替换

变量的名字是它的值保存的地方。引用它的值称为变量替换如果variable1是一个变量的名字，那么$variable1就是引用这个变量的值，即这个变量它包含的数据。

2.变量赋值

用“=”对变量进行赋值，“=”的两侧左右两边不能有空白符。

3.bash变量无类型

不同与许多其他的编程语言，bash不以"类型"来区分变量。本质上来说，bash变量是字符串，但是根据环境的不同，bash允许变量有整数计算和比较。其中的决定因素是变量的值是不是只含有数字。

示例3.2.2-1：对变量操作的脚本如下：

```
#!/bin/sh
a="hello world" #对变量赋值
echo "A is:" # 现在打印变量a的内容
echo $a
```

有时候变量名很容易与其他文字混淆，比如：

```
num=2
echo "this is the $numnd"
```

这并不会打印出"this is the 2nd"，而仅仅打印"this is the "，因为shell会去搜索变量numnd的值，但是这个变量时没有值的。可以使用花括号来告诉shell我们要打印的是num变量。

```
num=2
echo "this is the ${num}nd"
```

这将打印： this is the 2nd

5.位置参数

命令行传递给脚本的参数是: $0， $1， $2， $3 . . .

$0是脚本的名字，$1是第一个参数，$2是第二个参数，$3是第三个，以此类推。在位置参数$9之后的参数必须用括号括起来，例如：${10}， ${11}， ${12}。

特殊变量$*和$@ 表示所有的位置参数。

示例3.2.2-3:

位置参数实例:

```
#!/bin/sh
echo "number of vars:"$#
echo "values of vars:"$*
echo "value of var1:"$1
echo "value of var2:"$2
echo "value of var3:"$3
echo "value of var4:"$4
```

退出和退出状态

exit命令一般用于结束一个脚本，就像C语言的exit一样。它也能返回一个值给父进程。每一个命令都能返回一个退出状态(有时也看做返回状态)。

一个命令执行成功返回0，一个执行不成功的命令则返回一个非零值，此值通常可以被解释成一个对应的错误值。

同样的，在脚本里的函数和脚本自身都会返回一个退出状态码。在脚本或函数里被执行的最后一个命令将决定退出状态码。

如果一个脚本以不带参数的exit命令结束，脚本的退出状态码将会是执行exit命令前的最后一个命令的退出码。

脚本结束没有exit，不带参数的exit和exit $?三者是等价的。

03

流程控制

流程控制的主要目的是为脚本编程中的代码块进行控制和操作。

包括：条件测试、操作符、循环控制、分支控制等

3.3.1. 条件测试

每一个完善的编程语言都应该能测试一个条件。然后依据测试的结果做进一步的动作。bash有test命令，各种括号及内嵌的操作符，还有if/then结构来完成上面的功能。

大多数情况下，可以使用测试命令来对条件进行测试。比如可以比较字符串、判断文件是否存在及是否可读等等。通常用"[ ]"来表示条件测试。注意这里的空格很重要。要确保方括号两侧的空格。

1. 比较操作符

比较操作，包括整数比较操作，字符串比较操作和混合比较操作。其中，比较操作符如表3-2所示，字符串比较操作如表3-3所示，混和比较操作如表3-4所示。

表 3-2 常用整数比较操作

函数	说明
-eq	等于 if ["$a" -eq "$b"]
-ne	不等于 if ["$a" -ne "$b"]
-gt	大于 if ["$a" -gt "$b"]
-ge	大于等于 if ["$a" -ge "$b"]
-lt	小于 if ["$a" -lt "$b"]
-le	小于等于 if ["$a" -le "$b"]
<	小于(在双括号里使用) (("$a" < "$b"))
<=	小于等于 (在双括号里使用) (("$a" <= "$b"))
>	大于 (在双括号里使用) (("$a" > "$b"))
>=	大于等于(在双括号里使用) (("$a" >= "$b"))

1. 比较操作符

表 3-3 常用字符串比较操作

函数	说明
=	等于 if ["$a" = "$b"]
==	等于 if ["$a" == "$b"] 它和=是同义词。
!=	不相等 if ["$a" != "$b"]操作符在[[...]]结构里使用模式匹配.
<	小于，依照 ASCII 字符排列顺序 if [["$a" < "$b"]]，if ["$a" \< "$b"]注意"<"字符在[ ]结构里需要转义
>	大于，依照 ASCII 字符排列顺序 if [["$a" > "$b"]]，if ["$a" \> "$b"]注意">"字符在[ ]结构里需要转义.
-z	字符串为"null"，即是指字符串长度为零。
-n	字符串不为"null"，即长度不为零.

表 3-4 常用混合比较操作

函数	说明
-a	逻辑与，如果 exp1 和 exp2 都为真，则 exp1 -a exp2 返回真.
-o	逻辑或，只要 exp1 和 exp2 任何一个为真，则 exp1 -o exp2 返回真.

2. 文件测试操作符

表 3-5 常用文件测试操作

函数	说明
-e	文件存在
-f	文件是一个普通文件(不是一个目录或是一个设备文件)
-s	文件大小不为零
-d	文件是一个目录
-b	文件是一个块设备(软盘， 光驱， 等等.)
-c	文件是一个字符设备(键盘， 调制解调器， 声卡， 等等.)
-p	文件是一个管道
-h	文件是一个符号链接
-L	文件是一个符号链接
-S	文件是一个 socket
-t	文件(描述符)与一个终端设备相关
-r	文件是否可读 (指运行这个测试命令的用户的读权限)
-w	文件是否可写 (指运行这个测试命令的用户的读权限)
-x	文件是否可执行 (指运行这个测试命令的用户的读权限)
-g	文件或目录的设置-组-ID(sgid)标记被设置，如果一个目录的 sgid 标志被设置，在这个目录下创建的文件都属于拥有此目录的用户组，而不必是创建文件的用户所属的组。这个特性对在一个工作组里的同享目录很有用处。
-u	文件的设置-用户-ID(suid)标志被设置

分析下列测试命令含义

[[$a == z*]]	:	如果变量$a以字符"z"开始(模式匹配)则为真
[[$a == "z*"]]	:	如果变量$a与z*(字面上的匹配)相等则为真
[$a == z*]	:	文件扩展和单元分割有效.
["$a" == "z*"]	:	如果变量$a与z*(字面上的匹配)相等则为真
[-x "/bin/sh"]		判断/bin/sh是否存在并有可执行权限
[-r "somefile"]	:	判断文件是否可读
["$a" = "$b"]	:	判断$a和$b是否相等

3.3.2. 常用操作符

表 3-6 常用操作符

赋值操作符	=	通用的变量赋值操作符，可以用于数值和字符串的赋值
计算操作符	+	加
	-	减
	*	乘
	/	除
	**	求幂
	%	求模
位操作符	<<	位左移(每移一位相当乘以 2)
	<<=	"位左移赋值"
	>>	位右移(每移一位相当除以 2)
	>>=	"位右移赋值"(和<<=相反)
	&	位与
	&=	"位于赋值"
	|	位或
	|=	"位或赋值"
	~	位反
	!	位非
	^	位或
	^=	"位或赋值"
逻辑操作符	&&	逻辑与
	||	逻辑或

例3.3.2-1 用欧几里得算法计算最大公约数

欧几里得算法：

假设要找两个数 A 和 B 的最大公约数：

开始

↓

输入两个整数 A 和 B

↓

重复以下步骤直到余数为0：

1. 计算余数 r = A % B
2. A ← B
3. B ← r

↓

输出 A（最大公约数）

↓

结束

例3.3.2-1

```
#!/bin/bash
#最大公约数，使用 Euclid算法
#参数检测
ARGS=2
E_BADARGS=85
```

```
if [ $# -ne "$ARGS" ]
then
    echo "Usage: `basename $0` first-number second-number"
    exit $E_BADARGS
fi
```

```
gcd()
{
dividend=$1 #赋任意值
divisor=$2 #这里两个参数赋值大小有没有关系，为什么?
remainder=1
   #如果在循环中使用未初始化变量，在循环中第一个传递值会使他返回一个错误信息
   until [ "$remainder" -eq 0 ]
   do
   let "remainder = $dividend % $divisor"
   dividend=$divisor
   divisor=$remainder
   done
}
```

```
gcd $1 $2
echo
echo "GCD of $1 and $2 = $dividend"
echo
exit 0
```

3.3.3. 循环控制

对代码块的操作是构造组织shell脚本的关键，循环和分支结构为脚本编程提供了操作代码块的工具。

1. for

格式：

for arg in [list]

这是一个基本的循环结构，它与C的for结构有很大不同。

用法：

```
for arg in [list]
do
        command(s)...
done
```

Example3.3.3-1：分配行星的名字和它距太阳的距离

```
#!/bin/bash
for planet in "Mercury 36" "Venus 67" "Earth 93"  "Mars 142" "Jupiter 483"
do
    set -- $planet  # Parses variable "planet" and sets positional parameters.
    #"--" 将防止$planet为空,或者是以一个破折号开头。
    #可能需要保存原始的位置参数,因为它们被覆盖了。
    echo "$1              $2,000,000 miles from the sun"
    #-------two  tabs---把后边的0和$2连接起来
done
exit 0
```

2. while

这种结构在循环的开头判断条件是否满足，如果条件一直满足，那就一直循环下去(0为退出码[exit status])，与for 循环的区别是，这种结构适合用在循环次数未知的情况下。

```
while [condition]
do
        command...
done
```

和for循环一样，如果想把do和条件放到同一行上还是需要一个“；”。

```
while [condition] ;  do
```

示例3.3.3-2简单的while循环

```
#!/bin/bash
var0=0
LIMIT=10
while [ "$var0" -lt "$LIMIT" ]
do
    echo -n "$var0 "        # -n 将会阻止产生新行
    var0=`expr $var0 + 1`   # var0=$(($var0+1))  也可以
                            # var0=$((var0 + 1)) 也可以
                            # let "var0 += 1"    也可以
done                        # 使用其他的方法也行
echo
exit 0
```

3. until

这个结构在循环的顶部判断条件，并且如果条件一直为false那就一直循环下去(与while相反)。

```
until [condition-is-true]
do
        command...
done
```

until循环的判断在循环的顶部，这与某些编程语言是不同的。与for循环一样，如果想把do和条件放在一行里，就使用“；”。

```
until [condition-is-true] ;  do
```

示例3.3.3-3 until循环

```
#!/bin/bash
END_CONDITION=end
until [ "$var1" = "$END_CONDITION" ]#在循环的顶部判断条件.
do
echo "Input variable #1 "
echo "($END_CONDITION to exit)"
read var1
echo "variable #1 = $var1"
echo
done
exit 0
```

4. 影响循环行为的命令break，continue

break和continue这两个循环控制命令与其它语言的类似命令的行为是相同的，break命令将会跳出循环，continue命令将会跳过本次循环下边的语句，直接进入下次循环。

break命令可以带一个参数，一个不带参数的break循环只能退出最内层的循环，而break N可以退出N层循环。

continue命令也可以像break带一个参数，一个不带参数的continue命令只去掉本次循环的剩余代码。而continue N将会把N层循环剩余的代码都去掉，但是循环的次数不变。

3.3.4. 测试与分支

case和select结构在技术上说不是循环，因为它们并不对可执行的代码块进行迭代。但是和循环相似的是，它们也依靠在代码块的顶部或底部的条件判断来决定程序的分支。

在shell中的case同C/C++中的switch结构是相同的，它允许通过判断来选择代码块中多条路径中的一条。它的作用和多个if/then/else语句相同，是它们的简化结构，特别适用于创建目录。

1. case

```
case "$variable" in
?"$condition1" )
?command...
?; ;
?"$condition2" )
?command...
?; ;
esac
```

对变量使用""并不是强制的，因为不会发生单词分离。每句测试行，都以右小括号)结尾。每个条件块都以两个分号结尾。

case块的结束以esac(case的反向拼写)结尾。

示例3.3.4-1 用case查看计算机的架构

```
#!/bin/bash
#case-cmd.sh: 使用命令替换来产生"case"变量
case $( arch ) in   #arch"返回机器的类型,等价于'uname -m'...
    i386                )       echo "80386-based machine";;
    i486        )       echo "80486-based machine";;
    i586        )       echo "Pentium-based machine";;
    i686                )       echo "Pentium2+-based machine";;
x86_64)                 echo "inter10'cpu";;
*       )                echo "Other type of machine";;
esac
exit 0
```

2. select

select结构是建立菜单的另一种工具，从ksh中引入的结构如下：

```
select variable [in list]
do
    ?command...
    ?break
done
```

示例3.3.4-2用select来创建菜单

```
#!/bin/bash
PS3='Choose your favorite vegetable: ' #PS3 是 Bash select 特有的环境变量，用于设置提示符字串
echo
select vegetable in "beans" "carrots" "potatoes" "onions" "rutabagas"
do
    echo
    echo "Your favorite veggie is $vegetable."
    echo "Yuck!"
    echo
    break  # 如果这里没有'break'会发生什么?
done
exit 0
```

重庆理工大学/电气学院

CHONGQING UNIVERSITY OF TECHNOLOGY

嵌入式Linux系统开发教程

—基于ARM处理器通用平台（arm9-arm11-contexA系列）