华 清 远 见
FARSIGHT
嵌 入 式 培 训 专 家

嵌入式学院
华清远见旗下品牌

Shell编程基础

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell测试语句
4. Shell控制语句
5. Shell函数
6. Shell 编程调试方法

Shell脚本的本质

嵌入式学院
华清远见旗下品牌

- 编译型语言
- 解释型语言
  - Shell脚本语言是解释型语言
  - Shell脚本的本质：
    - **Shell命令的有序集合**。

shell 编程的基本过程

嵌入式学院
华清远见旗下品牌

基本过程分为三步:

Step1. 建立 shell 文件

包含任意多行操作系统命令或shell命令的文本文件;

Step2. 赋予shell文件执行权限

用chmod命令修改权限;

Step3. 执行shell文件

直接在命令行上调用shell程序.

实例

嵌入式学院
华清远见旗下品牌

Step1: 建立shell文件 (可用任何建立文本文件的方法)

```
$ cat  prog1.sh
 date
```

Step2: 赋予执行权限：(初始文本文件无执行权限)

```
$ chmod  740  prog1.sh
```

Step3: 执行该shell程序

```
$ prog1.sh
prog1.sh: not found
(shell在标准搜索路径中找不到prog1.sh)
```

Step4: 指定路径或修改环境变量PATH后执行shell程序

```
$ ./prog1.sh
2009年  12月  16日  星期二  14:52:57  CST
```

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell测试语句
4. Shell控制语句
5. Shell函数
6. Shell 编程调试方法

Shell变量

嵌入式学院
华清远见旗下品牌

- Shell允许用户建立变量存储数据，但不支持数据类型（整型、字符、浮点型），将任何赋给变量的值都解释为一串字符

Variable=value

- 命名规则同C++中的命名规则
- count=1
- echo $count
- DATE=`date`
- echo $DATE

shell变量

嵌入式学院
华清远见旗下品牌

- Bourne Shell有如下四种变量：
  - 用户自定义变量
  - 位置变量即 命令行参数
  - 预定义变量
  - 环境变量

用户自定义变量

嵌入式学院
华清远见旗下品牌

- 在**shell**编程中通常使用全大写变量，方便识别
  - $ COUNT=1
- 变量的调用：在变量前加**$**
  - $ echo $HOME
- **Linux Shell/bash**从右向左赋值
  - $Y=y
  - $ X=$Y
  - $ echo $X
  - y
- 使用**unset**命令删除变量的赋值
  - $ Z=hello
  - $ echo $Z
  - hello
  - $ unset Z
  - $ echo $Z

位置变量

嵌入式学院
华清远见旗下品牌

- **$0** 与键入的命令行一样，包含脚本文件名
- **$1,$2,……$9** 分别包含第一个到第九个命令行参数
- **$#** 包含命令行参数的个数
- **$@** 包含所有命令行参数：“$1,$2,……$9”
- **$?** 包含前一个命令的退出状态
- **$*** 包含所有命令行参数：“$1,$2,……$9”
- **$$** 包含正在执行进程的ID号

常用shell环境变量

嵌入式学院
华清远见旗下品牌

- **HOME**： /etc/passwd文件中列出的用户主目录
- **IFS**：Internal Field Separator, 默认为空格，tab及换行符
- **PATH** ： shell搜索路径
- **PS1**，**PS2**：默认提示符($)及换行提示符(>)
- **TERM**：终端类型，常用的有vt100,ansi,vt200,xterm等

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell功能语句
4. Shell控制语句
5. Shell函数
6. Shell 编程调试方法

shell 程序和语句

shell 程序由零或多条shell语句构成。 shell语句包括三类:说明性语句、功能性语句和结构性语句。

说明性语句:

以#号开始到该行结束，不被解释执行

功能性语句:

任意的shell命令、用户程序或其它shell程序。

结构性语句:

条件测试语句、多路分支语句、循环语句、循环控制语句等。

说明性语句(注释行)

嵌入式学院
华清远见旗下品牌

注释行可以出现在程序中的任何位置,既可以单独占用一行，也可以接在执行语句的后面. 以#号开始到所在行的行尾部分，都不被解释执行. 例如:

```
#! /bin/sh
#
# 本程序说明
#
command_1
command_2              # command_2的语句说明
……
# 下面程序段的说明
command_m
……
command_n              # commandn_语句的说明
……
```

告诉OS用哪种类型的shell来解释执行该程序

常用功能性语句(命令)

嵌入式学院
华清远见旗下品牌

read

read从标准输入读入一行, 并赋值给后面的变量,其语法为:

. **read** var

把读入的数据全部赋给var

. **read** var1 var2 var3

把读入行中的第一个单词(word)赋给var1, 第二个单词赋给var2， ……把其余所有的词赋给最后一个变量.

如果执行read语句时标准输入无数据, 则程序在此停留等侯, 直到数据的到来或被终止运行。

应用实例


```
# example1  for  read
echo  "Input  your  name: \c"
read  username
echo  "Your name is  $username"

#example2  for  read
echo "Input  date  with  format  yyyy  mm dd: \c"
read  year  month  day
echo  "Today  is  $year/$month/$day,  right?"
echo  "Press  enter  to  confirm  and  continue\c"
read  answer
echo "I  know  the  date,  bye!"
```

expr

算术运算命令expr主要用于进行简单的整数运算，包括加(+)、减（-）、乘（*)、整除（/）和求模（%）等操作。例如:

```
$ expr  12  +  5  \*  3
27
$ expr  3  -  8  /  2
-1

$ num=9
$ sum=`expr  $num  \*  6 `
$ echo  $sum
54
```

反斜线去掉*号的元字符含义

反撇号引用命令的运行结果

test

test语句可测试三种对象:

字符串　　整数　　文件属性

每种测试对象都有若干测试操作符

例如:

test "$answer" = "yes"

变量answer的值是否为字符串yes

test $num –eq 18

变量num的值是否为整数18

test -d tmp

测试tmp是否为一个目录名

字符串测试

- s1 = s2　　测试两个字符串的内容是否完全一样
- s1 != s2　　测试两个字符串的内容是否有差异
- -z s1　　测试s1 字符串的长度是否为0
- -n s1　　测试s1 字符串的长度是否不为0

整数测试

- a -eq b　　测试a 与b 是否相等
- a -ne b　　测试a 与b 是否不相等
- a -gt b　　测试a 是否大于b
- a -ge b　　测试a 是否大于等于b
- a -lt b　　测试a 是否小于b
- a -le b　　测试a 是否小于等于b

文件测试

嵌入式学院
华清远见旗下品牌

-d name	测试name 是否为一个目录
-f name	测试name 是否为普通文件
-L name	测试name 是否为符号链接
-r name	测试name 文件是否存在且为可读
-w name	测试name 文件是否存在且为可写
-x name	测试name 文件是否存在且为可执行
-s name	测试name 文件是否存在且其长度不为0
f1 -nt f2	测试文件f1 是否比文件f2 更新
f1 -ot f2	测试文件f1 是否比文件f2 更旧

tput

tput命令主要用于设置终端工作模式，或读出终端控制字符．tput命令与终端控制代码数据库terminfo相连，根据shell环境变量TERM的值，读出这种终端的指定功能控制代码．常用的终端功能控制如下表：

选项	功　能	选项	功　能
bel	终端响铃	el	光标位置到行末清字符
blink	闪烁显示	smso	启动突出显示模式
bold	粗体字显示	smul	启动下划线模式
clear	清屏	rmso	结束突出显示模式
cup r c	光标移到 r 行 c 列	rmul	结束下划线模式
dim	显示变暗	rev	反白显示
ed	光标位置到屏幕底清屏	sgr0	关闭所有属性

应用实例一:

```
tput  clear
tput  cup  11  23
tput  rev
echo  "Hello,  everybody!"
tput  sgr0
tput  cup  24  1
```

该程序先清屏，并在屏幕中央位置(11行23列)用反极性显示字符串“Hello, everybody!”，恢复正常显示极性后光标停留在屏幕左下角。

应用实例二

```
# program2   for   tput
#
bell=`tput  bel`
s_uline=`tput  smul`
e_uline=`tput  rmul`
tput  clear
tput  cup  10  20
echo  $bell   $s_uline
echo  "Computer  Department"
echo  $e_uline
```

注意
在$bell和$s_uline之间的空格将同样在屏幕上显示，使Computer Department的实际位置向右移动一格

功能：响一声铃后，在清空的屏幕中央以下划线模式显示字符串
"Computer Department"

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell功能语句
4. Shell控制语句
5. Shell函数
6. Shell 编程调试方法

结构性语句

结构性语句主要根据程序的运行状态、输入数据、变量的取值、控制信号以及运行时间等因素来控制程序的运行流程。

主要包括：条件测试语句（两路分支）、多路分支语句、循环语句、循环控制语句和后台执行语句等。

条件语句

if...then...fi

语法结构:

```
if   表达式
        then  命令表
fi
```

如果表达式为真, 则执行命令表中的命令; 否则退出if语句, 即执行fi后面的语句。

if和fi是条件语句的语句括号, 必须成对使用;

命令表中的命令可以是一条, 也可以是若干条。

实例

嵌入式学院
华清远见旗下品牌

shell程序prog2.sh（测试命令行参数是否为已存在的文件或目录）。用法为：

./prog2.sh file

代码如下：

```
#The statement of  if…then…fi                (注释语句)
if  [  -f  $1  ]       (测试参数是否为文件)
then
   echo "File  $1  exists"                  (引用变量值)
fi
if  [  -d  $HOME/$1  ]                  (测试参数是否为目录)
then
   echo "File  $1 is  a  directory"       (引用变量值)
fi
```

执行prog2程序:

$ ./prog2.sh prog1.sh

File prog1.sh exists

$0为prog2.sh; $1为prog1.sh, 是一个已存在的文件.

$ ./prog2.sh backup

File backup is a directory

$0为prog2.sh; $1为backup,是一个已存在的目录.

如果不带参数, 或大于一个参数运行prog2, 例如:

$ ./prog2.sh (或 $ ./prog2.sh file1 file2)

会出现什么结果?

条件语句

if...then...else...fi

语法结构为:

```
if    表达式
then 命令表1
else  命令表2
fi
```

如果表达式为真, 则执行命令表1中的命令, 再退出if语句; 否则执行命令表2中的语句, 再退出if语句.

注意: 无论表达式是否为真, 都有语句要执行.

test命令的使用

test命令测试的条件成立时, 命令返回值为真(0)，否则返回值为假(非0).

方式1

```
test  $name  -eq  $1
echo  $?
```

方式2

```
if  test  -f  $filename
then
    ……
fi
```

方式3

```
if  [  -f  $filename  ]
then
    ……
fi
```

用方括号替代test语句

至少一个空格

至少一个空格

例子:　shell程序prog3.sh, 用法为:

./prog3.sh　file

内容如下:

```
#The statement of if…then…else…fi
if  [  -d  $1  ]
then
        echo "$1  is  a  directory"
        exit        (退出当前的shell程序)
else
        if  [  -f   $1  ]
        then
                echo  "$1  is  a  common  file"
        else
                echo  "unknown"
        fi
fi
```

运行prog3.sh程序:

```
$ ./prog3.sh  backup
backup  is  a  directory

$ ./prog3.sh  prog1
prog1  is  a  common  file

$ ./prog3.sh  abc
unknown
```

prog3.sh是对prog2.sh的优化, 逻辑结构更加清晰合理!

多路分支语句

嵌入式学院
华清远见旗下品牌

case...esac

多路分支语句case用于多重条件测试, 语法结构清晰自然. 其语法为:

```
case  字符串变量  in
    模式1)
        命令表1
        ;;
    模式2)
        命令表2
        ;;
    ……
    模式n)
        命令表n
        ;;
esac
```

case语句只能检测字符串变量

各模式中可用文件名元字符,以右括号结束

命令表以单独的双分号行结束,退出case语句

模式 n常写为字符* 表示所有其它模式

最后一个双分号行可以省略

实例. 程序prog4.sh检查用户输入的文件名,用法为:

```
        ./prog4.sh    string_name
# The statement of  case…esac
if  [  $#  -eq  0  ]
then
        echo  "No argument is declared"
        exit
fi
case  $1  in
        file1)
                echo  "User selects file1"
                ;;
        file2)
                echo  "User selects file2"
                ;;
        *)
                echo  "You must select either file1 or file2!"
                ;;
esac
```

循环语句

嵌入式学院
华清远见旗下品牌

for...do...done

当循环次数已知或确定时，使用for循环语句来多次执行一条或一组命令。 循环体由语句括号do和done来限定。格式为:

for 变量名 in 单词表

do

命令表

done

变量依次取单词表中的各个单词，每取一次单词，就执行一次循环体中的命令. 循环次数由单词表中的单词数确定. 命令表中的命令可以是一条，也可以是由分号或换行符分开的多条。

如果单词表是命令行上的所有位置参数时，可以在for语句中省略 “in 单词表” 部分。

实例

程序prog5.sh拷贝当前目录下的所有文件到backup子目录下. 使用语法为: ./prog5.sh [filename]

实例

嵌入式学院
华清远见旗下品牌

```
# The statement of for…do…done
if [ ! -d $HOME/backup ]
then
    mkdir $HOME/backup
fi
flist=`ls`
for file in $flist
do
    if [ $# = 1 ]
    then
        if [ $1 = $file ]
        then
            echo "$file found" ; exit
        fi
    else
        cp $file $HOME/backup
        echo "$file copied"
    fi
done
echo ***Backup Completed***
```

flist的值是ls的执行结果即当前目录下的文件名

命令行上有一个参数时

命令行上不带参数时

循环语句

while...do...done

语法结构为：

```
while  命令或表达式
do
        命令表
done
```

while语句首先测试其后的命令或表达式的值，如果为真，就执行一次循环体中的命令，然后再测试该命令或表达式的值，执行循环体，直到该命令或表达式为假时退出循环。

while语句的退出状态为命令表中被执行的最后一条命令的退出状态。

实例

创建文件程序prog6，批量生成空白文件，用法为:

prog6 file [number] ./a.sh file 6

```
# The statement for while
if [ $# = 2 ]
then
        loop=$2
else
        loop=5
fi
i=1
while [ $i -lt $loop ]
do
        > $1$i
        i=`expr $i + 1`
done
```

根据命令行的第二个参数来确定循环的次数

建立以第一个参数指定的文件名前缀，例如以“file”开头, 变量i的值结尾的空文件名. 参见命令cmd > file

until...do...done

语法结构为：　until　命令或表达式

do

命令表

done

until循环与while循环的功能相似，所不同的是只有当测试的命令或表达式的值是假时，才执行循环体中的命令表，否则退出循环．这一点与while命令正好相反．

循环控制语句

嵌入式学院
华清远见旗下品牌

break 和 continue

break n 则跳出n层;

continue语句则马上转到最近一层循环语句的下一轮循环上,

continue n则转到最近n层循环语句的下一轮循环上.

实例. 程序prog7的用法为:

prog7 整数 整数 整数 …

参数个数不确定, 范围为1~10个, 每个参数都是正整数。

```
if [ $# = 0 ]
then
        echo "Numeric arguments required"
        exit
fi
if [ $# -gt 10 ]
then
        echo "Only ten arguments allowed"
        exit
fi
for number
do
        count=`expr $number % 2`
        if [ $count -eq 1 ]
        then
                continue
        else
                output="$output $number"
        fi
done
echo "Even numbers: $output "
```

取所有位置参数

用2求模, count的值只能是0或1

下一轮循环

是奇数

偶数放到偶数队列中

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell测试语句
4. Shell控制语句
5. Shell函数
6. Shell 编程调试方法

shell 函数

在shell程序中, 常常把完成固定功能、且多次使用的一组命令（语句）封装在一个函数里，每当要使用该功能时只需调用该函数名即可。

函数在调用前必须先定义，即在顺序上函数说明必须放在调用程序的前面。

调用程序可传递参数给函数, 函数可用return语句把运行结果返回给调用程序。

函数只在当前shell中起作用, 不能输出到子Shell中。

shell 函数

函数定义格式:

方式一:

```
function_name ( )
{
        command1
        ……
        commandn
}
```

方式二:

```
function  function_name ( )
{
        command1
        ……
        commandn
}
```

shell 函数调用

嵌入式学院
华清远见旗下品牌

函数调用格式:

函数的所有标准输出都传递给了主程序的变量

方式**1**:

```
value_name=`function_name [arg1 arg2 … ]`
```

获取函数的返回的状态

方式**2**:

```
function_name [arg1 arg2 … ]
echo $?
```

实例


```
check_user( )     #查找已登录的指定用户
{
        user=`who  |  grep  $1 | wc -l`
        if [ $user –eq 0 ]
        then
              return  0       #未找到指定用户
        else
              return  1       #找到指定用户
        fi
}
while  true          # MAIN, Main, main:   program  begin  here
do
              echo  "Input username: \c"
              read   uname
              check_user  $uname       # 调用函数, 并传递参数uname
              if [ $? –eq  1 ]              # $?为函数返回值
              then        echo  "user  $uname  online"
              else        echo  "user  $uname  offline"
              fi
done
```

函数变量作用域

嵌入式学院
华清远见旗下品牌

- 全局作用域：在脚本的其他任何地方都能够访问该变量。
- 局部作用域：只能在声明变量的作用域内访问。
- 声明局部变量的格式：
  - ***Local*** **variable_name =value**

函数变量作用域

嵌入式学院
华清远见旗下品牌

- Scope()

```
{
Local lclvariable =1
Gblvariable = 2
echo “lclavariable in function = $ lclvariable ”
echo “Gblvariable in function = $ Gblvariable ”
}

Scope

echo “lclavariable in function = $ lclvariable ”
echo “Gblvariable in function = $ Gblvariable ”
```

内容提纲

嵌入式学院
华清远见旗下品牌

1. Shell基础
2. Shell变量及环境变量
3. Shell测试语句
4. Shell控制语句
5. Shell函数
6. Shell 脚本调试方法

shell 编程的调试方法

嵌入式学院
华清远见旗下品牌

- 脚本调试的主要工作就是发现引发脚本错误的原因以及在脚本源代码中定位发生错误的行，常用的手段包括：
  - 分析输出的错误信息。
  - 通过在脚本中加入调试语句，输出调试信息来辅助诊断错误。
  - 利用调试工具等。

进行Shell语法检查


```
#!/bin/bash
isRoot()
{
    if [ "$UID" -ne 0 ]
            return 1
    else
            return 0
    fi
 }

isRoot

if  [ "$?" -ne 0 ]
then
     echo "Must be root to run this script"
     exit 1
else
    echo "welcome root user“
     #do something
fi
```

- *进行**Shell**语法检查*
- *之后运行：* ***sh –n prog8.sh***

或 sh –vn prog8.sh

跟踪脚本执行结果

- 在希望开始调试的地方插入 set –x,
- 在希望结束调试的地方插入 set +x

  - set –x
  - echo “Guess the secret color”
  - set +x

•跟踪程序的每一步的执行结果

Q&A