嵌入式系统工程师

Linux下编程工具(shell脚本)

大纲

- linux环境开发概述
- linux文件及目录结构
- linux常用命令
- linux文本编辑器vi+gedit
- linuxshell脚本编程
- linux编译器gcc
- linux调试器gdb
- linux工程管理软件—make

大纲

- linux环境开发概述
- linux文件及目录结构
- linux常用命令
- linux文本编辑器vi+gedit
- **linuxshell脚本编程**
- linux编译器gcc
- linux调试器gdb
- linux工程管理软件—make

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
  - 条件测试语句
  - 控制语句
  - 函数
  - 系统shell启动顺序

- shell的两层含义:

  既是一种应用程序,又是一种程序设计语言

  - **作为应用程序:**

    交互式地解释、执行用户输入的命令，将用户的操作翻译成机器可以识别的语言，完成相应功能

  - **作为程序设计语言:**

    它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支

    完成类似于windows下批处理操作，简化我们对系统的管理与应用程序的部署

Linux shell脚本编程

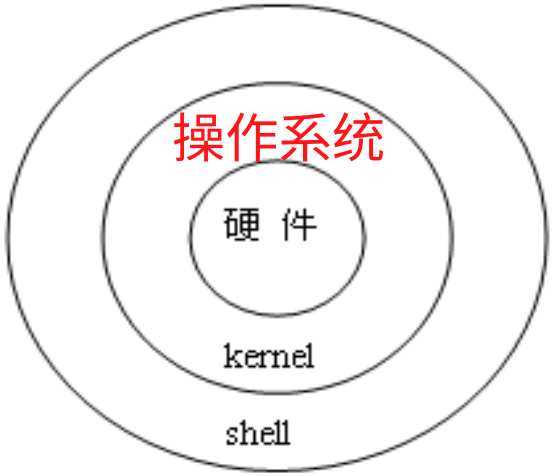
- 作为应用程序：
  - 称之为shell命令解析器

    shell是用户和Linux内核之间的接口程序

    用户在提示符下输入的命令都由shell先解释然后传给Linux核心

    它调用了系统核心的大部分功能来执行程序、并以并行的方式协调各个程序的运行

- Linux系统中提供了好几种不同的shell命令解释器，如sh、ash、bash等。
- 一般默认使用bash作为默认的解释器
- **我们后面编写的shell脚本，都是由上述shell命令解释器解释执行的。**



Linux shell脚本编程

- 作为程序设计语言
  - 称之为**shell脚本**
    - 我们学过的c/c++等语言，属于编译性语言（编写完成后需要使用编译器完成编译、汇编、链接等过程变为二进制代码方可执行）
    - shell脚本是一种脚本语言，我们只需使用任意文本编辑器，按照语法编写相应程序，增加可执行权限，即可在安装shell命令解释器的环境下执行
  - shell脚本主要用于：
    - 帮助开发人员或系统管理员将复杂而又反复的操作放在一个文件中，通过简单的一步执行操作完成相应任务，从而解放他们的负担

- shell应用举例:

1、《linux常用命令_练习.txt》

我们前面完成了这个练习，步骤很多，其实我们只需要将所有操作写入一个文件——cmd.sh(名字跟后缀可任取，为了便于区分我们一般写为*.sh形式)

然后:

```
chmod +x cmd.sh
./cmd.sh直接执行即可
```

2、假设我们要完成以下任务:

判断用户家目录下（~）下面有没有一个叫test的文件夹

如果没有，提示按y创建并进入此文件夹，按n退出

如果有，直接进入，提示请输入一个字符串，并按此字符串创建一个文件，如果此文件已存在，提示重新输入，重复三次自动退出，不存在创建完毕，退出

简单的进行命令堆积无法完成以上任务，这就需要学习相应的shell脚本语法规则了

Linux shell脚本编程

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
  - 条件测试语句
  - 控制语句
  - 函数

Linux shell脚本编程

➢shell脚本的定义与执行:

1、定义以开头: #!/bin/sh

#!用来声明脚本由什么shell解释，否则使用默认shell

2、单个"#"号代表注释当前行

3、执行:

#chmod +x test.sh ./test.sh 增加可执行权限后执行

#bash test.sh 直接指定使用bash解释test.sh

#. test.sh(source test.sh) 调用默认shell解释test.sh

例: 1.sh

```
1 #!/bin/bash
2 clear
3 echo "this is the first shell script"
```

```
#chmod +x test.sh;./test.sh
#bash test.sh
#. test.sh
```

- 三种执行脚本的方式不同点:
  - ./和bash执行过程基本一致，后者明确指定bash解释器去执行脚本，脚本中#!指定的解释器不起作用

    前者首先检测#!，使用#!指定的shell，如果没有使用默认的shell
  - 用./和bash去执行会在后台启动一个新的shell去执行脚本

    用.去执行脚本不会启动新的shell,直接由当前的shell去解释执行脚本。

Linux shell脚本编程

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
    - 自定义变量、环境变量、预设变量
    - "" ' `` \ () {}
  - 条件测试语句
  - 控制语句
  - 函数

shell脚本变量(1/11)

- 自定义变量
  - 定义变量

    变量名=变量值

    如: num=10
  - 引用变量

    $变量名

    如: i=$num　　把变量num的值付给变量i
  - 显示变量

    使用echo命令可以显示单个变量取值

    echo $num
  - 清除变量

    使用unset命令清除变量

    unset varname

shell脚本变量(2/11)

- 变量的其它用法:
  - read string

    从键盘输入一个字符串付给变量string，若string没定义则先定义在使用
  - readonly var=100

    定义一个只读变量,只能在定义时初始化,以后不能改变,不能被清除。
  - export var=300

    使用export说明的变量，会被导出为环境变量，其它shell均可使用

    注意：此时必须使用source 2_var.sh才可以生效

shell脚本变量(3/11)

- 注意事项:
  - 1、变量名只能英文字母数字下划线，不能以数字开头

    1_num=10　错误

    num_1=20　正确
  - 2、等号两边不能直接接空格符，若变量中本身就包含了空格，则整个字符串都要用双引号、或单引号括起来；双引号内的特殊字符可以保有变量特性，但是单引号内的特殊字符则仅为一般字符。

```
name=aa bb                    //错误
name="aa bb"                  //正确
echo "$name is me"            //输出: aa bb is me
echo '$name is me'            //输出: $name is me
```

shell脚本变量(4/11)

例: 2_var.sh

```
1  #!/bin/bash
2  echo "this is the var test shell script "
3  name="sunplus edu"
4  echo "$name is me"
5  echo '$name is me'
6
7  echo "please input a string"
8  read string
9  echo "the string of your input is $string"
10
11 readonly var=1000
12 #var=200
13
14 export public_var=300
```

关了终端就没有了

shell脚本变量(5/11)

- 环境变量:
  - shell在开始执行时就已经定义了一些和系统的工作环境有关的变量，我们在shell中可以直接使用$name引用
  - 定义:

    一般在~/.bashrc或/etc/profile文件中（系统自动调用的脚本）使用export设置，允许用户后来更改

    VARNAME = value ; export VARNAME
  - 传统上，所有环境变量均为大写
  - 显示环境变量

    使用env 命令可以查看所有的环境变量。
  - 清除环境变量

    使用unset 命令清除环境变量

shell脚本变量(6/11)

- 常见环境变量:
  - HOME: 用于保存注册目录的完全路径名。
  - PATH: 用于保存用冒号分隔的目录路径名，shell将按PATH变量中给出的顺序搜索这些目录，找到的第一个与命令名称一致的可执行文件将被执行。

    PATH=$HOME/bin:/bin:/usr/bin;export PATH
  - HOSTNAME: 主机名
  - SHELL: 默认的shell命令解析器
  - LOGNAME: 此变量保存登录名
  - PWD: 当前工作目录的绝对路径名
  - ……

shell脚本变量(7/11)

例: 3_export.sh

```
1 #!/bin/bash
2
3 echo "You are welcome to use bash"
4 echo "Current work dirctory is $PWD"
5 echo "the host name is  $HOSTNAME"
6 echo "your home dir  $HOME"
7 echo "Your shell is  $SHELL"
```

shell脚本变量(8/11)

- 预定义变量:

  $#: 传给shell脚本参数的数量

  $*: 传给shell脚本参数的内容

  $1、$2、$3、...、$9: 运行脚本时传递给其的参数，用空格隔开

  $?: 命令执行后返回的状态

  "$?"用于检查上一个命令执行是否正确(在Linux中，命令退出状态为0表示该命令正确执行，任何非0值表示命令出错)。

  $0: 当前执行的进程名

  $$: 当前进程的进程号

  "$$"变量最常见的用途是用作临时文件的名字以保证临时文件不会重复

shell脚本变量(9/11)

例：4_$.sh

```
1  #!/bin/bash
2  echo "your shell script name is $0"
3  echo "the params of your input is $*"
4  echo "the num of your input  params is $#"
5  echo "the params is $1 $2 $3 $4"
6
7  ls
8  echo "the cmd state is $?"
9  cd /root
10 echo "the cmd state is $?"
11
12 echo "process id is $$"
```

shell脚本变量(10/11)

- 脚本变量的特殊用法: "" `` ' \ () {}
  - ""（双引号）：包含的变量会被解释
  - ''（单引号）：包含的变量会当做字符串解释
  - ``(数字键1左面的反引号)：反引号中的内容作为系统命令，并执行其内容，可以替换输出为一个变量

    ```
    $ echo "today is `date` "
    today is 2012年07月29日星期日 12:55:21 CST
    ```
  - \ 转义字符:

    同c语言 \n \t \r \a等 echo命令需加-e转义
  - (命令序列):

    由子shell来完成,不影响当前shell中的变量
  - { 命令序列 }:

    在当前shell中执行，会影响当前变量

shell脚本变量(11/11)

- 例:
  5_var_spe.sh

注意:

“{”、“}”
前后有一空格

```
1  #!/bin/bash
2  name=teacher
3  string1="good moring $name"
4  string2='good moring $name'
5  echo $string1
6  echo $string2
7
8  echo "today is `date`  "
9  echo 'today is `date`  '
10
11 echo -e "this \n is\ta\ntest"
12
13 ( name=student;echo "1 $name" )
14 echo 1:$name
15 { name=student; echo "2 $name";  }
16 echo 2:$name
```

Linux shell脚本编程

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
  - 条件测试语句
    - 文件、字符串、数字、复合测试
  - 控制语句
  - 函数

条件测试(1/7)

- 在写shell脚本时，经常遇到的问题就是判断字符串是否相等，可能还要检查文件状态或进行数字测试，只有这些测试完成才能做下一步动作
- test命令：用于测试字符串、文件状态和数字
- test命令有两种格式：

  test condition　或[condition]

  使用方括号时，要注意在条件两边加上空格
- shell脚本中的条件测试如下：

  文件测试、字符串测试、数字测试、复合测试
- 测试语句一般与后面讲的条件语句联合使用

条件测试(2/7)

- 文件测试：测试文件状态的条件表达式

  -e 是否存在　　-d 是目录　　-f 是文件

  -r 可读　　-w 可写　　-x 可执行

  -L 符号连接　　-c是否字符设备　　-b是否块设备

  -s 文件非空

例：6_test_file.sh

```
#!/bin/bash
test -e /dev/qaz
echo $?

test -e /home
echo $?

test -d /home
echo $?

test -f /home
echo $?

mkdir test_sh
chmod 500 test_sh

[ -r test_sh ]
echo $?

[ -w test_sh ]
echo $?

[ -x test_sh ]
echo $?

[ -s test_sh ]
echo $?

[ -c /dev/console ]
echo $?
[ -b /dev/sda ]
echo $?
[ -L /dev/stdin ]
echo $?
```

条件测试(3/7)

- 字符串测试:

  test str_operator “str”

  test “str1” str_operator “str2”

  [str_operator “str”]

  [“str1” str_operator “str2”]

- 其中str_operator可以是:

  = 两个字符串相等　　!= 两个字符串不相等

  -z 空串　　-n 非空串

条件测试(4/7)

例：7_test_string.sh

```
1  #!/bin/bash
2  test -z $yn
3  echo $?
4
5  echo "please input a y/n"
6  read yn
7  [ -z "$yn" ]
8  echo 1:$?
9
10 [ $yn = "y" ]
11 echo 2:$?
```

条件测试（5/7）

- 测试数值格式如下:

  test num1 num_operator num2

  [num1 num_operator num2]

- num_operator可以是:

  -eq 数值相等

  -ne 数值不相等

  -gt 数1大于数2

  -ge 数1大于等于数2

  -le 数1小于等于数2

  -lt 数1小于数2

例: 8_test_num.sh

```
1  #!/bin/bash
2  echo "please input a num(1-9)"
3  read num
4  [ $num -eq 5 ]
5  echo $?
6  [ $num -ne 5 ]
7  echo $?
8  [ $num -gt 5 ]
9  echo $?
10 [ $num -ge 5 ]
11 echo $?
12 [ $num -le 5 ]
13 echo $?
14 [ $num -lt 5 ]
15 echo $?
```

条件测试（6/7）

- 命令执行控制:
  - &&:

    command1 && command2

    &&左边命令（command1）执行成功(即返回0）shell才执行&&右边的命令（command2）
  - ||

    command1 || command2

    ||左边的命令（command1）未执行成功(即返回非0）shell才执行||右边的命令（command2）

例:

```
test -e /home && test -d /home && echo "true"
test 2 -lt 3 && test 5 -gt 3 && echo "equal"
test "aaa"="aaa" || [ "$yn" = "y" ]&& echo "equal"
```

Linux shell脚本编程

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
  - 条件测试语句
  - 控制语句
    - if case for while until break
  - 函数

控制结构(2/14)

- if [条件1]; then
  执行第一段程序
  else
  执行第二段程序
  fi

- if [条件1]; then
  执行第一段程序
  elif [条件2]; then
  执行第二段程序
  else
  执行第三段程序
  fi

控制结构(3/14)

例：9_if_then.sh

```
1 #!/bin/bash
2 echo "Press y to continue"
3 read yn
4 if [ $yn = "y" ]; then
5     echo "script is running..."
6 else
7     echo "stopped!"
8 fi
```

控制结构(4/14)

```
case $变量名称 in
    “第一个变量内容”）
        程序段一
        ;;
    “第二个变量内容”）
        程序段二
        ;;
    *)
        其它程序段
        exit 1
esac
```

控制结构(5/14)

例: 10_case1.sh 10_case2.sh

```
#!/bin/bash
echo "This script will print your choice"

case "$1" in
    "one")
        echo "your choice is one"
        ;;
    "two")
        echo "your choice is two"
        ;;
    "three")
        echo "Your choice is three"
        ;;
    *)
        echo "Error Please try again!"
        exit 1
        ;;
esac
```

```
#!/bin/bash

echo "Please input your choice:"
read choice

case "$choice" in
    Y | yes | Yes | YES)
    echo "It's right"
    ;;
    N* | n*)
    echo "It's wrong"
    ;;
    *)
    exit 1
esac


```

控制结构(6/14)

- for （(初始值; 限制值; 执行步阶 ））
  do
  　　程序段
  done
  初始值: 变量在循环中的起始值
  限制值: 当变量值在这个限制范围内时，就继续进行循环
  执行步阶: 每作一次循环时，变量的变化量

控制结构(7/14)

- declare是bash的一个内建命令，可以用来声明shell变量、设置变量的属性。declare也可以写作typeset。
- declare -i s 代表强制把s变量当做int型参数运算。

例:

11_for1.sh

```
1 #!/bin/bash
2 declare -i sum
3 for (( i=1; i<=100; i=i+1 ))
4 do
5       sum=sum+i
6 done
7
8 echo "The result is $sum"
```

控制结构(8/14)

- for var in con1 con2 con3 ...
  do
  　　程序段
  done
  第一次循环时，$var的内容为con1
  第二次循环时，$var的内容为con2
  第三次循环时，$var的内容为con3
  ……

控制结构(9/14)

例：11_for2.sh

```
1 #!/bin/bash
2 for i in 1 2 3 4 5 6 7 8 9
3 do
4     echo $i
5 done
```

例：11_for3.sh

```
1  #!/bin/bash
2  for name in `ls`
3  do
4  if [ -f $name ];then
5      echo "$name is file"
6  elif [ -d $name ];then
7      echo "$name is directory"
8  else
9      echo "^_^"
10 fi
11 done
```

控制结构(10/14)

- while [condition]

  do

  　　程序段

  done

  当condition成立的时候进入while循环，直到condition不成立时才退出循环。

控制结构(11/14)

例：12_while.sh

```
1 #!/bin/bash
2 declare -i i
3 declare -i s
4 while [ "$i" != "101" ]
5 do
6       s+=i;
7       i=i+1;
8 done
9 echo "The count is $s"
```

控制结构(12/14)

- until [condition]

  do

  　　程序段

  done

这种方式与while恰恰相反，当condition成立的时候退出循环，否则继续循环。

控制结构(13/14)

例：13_until.sh

```
1 #!/bin/bash
2 declare -i i
3 declare -i s
4 until [ "$i" = "101" ]
5 do
6       s+=i;
7       i=i+1;
8 done
9 echo "The count is $s"
```

控制结构(14/14)

- break
  - break命令允许跳出循环。
  - break 通常在进行一些处理后退出循环或case 语句
- continue
  - continue 命令类似于break 命令
  - 只有一点重要差别，它不会跳出循环，只是跳过这个循环步

Linux shell脚本编程

- shell概述
- shell语法
  - shell脚本的定义与执行
  - 变量
  - 条件测试语句
  - 控制语句
  - 函数
    - 函数的定义与使用

函数（1/3）

- 有些脚本段间互相重复，如果能只写一次代码块而在任何地方都能引用那就提高了代码的可重用性。
- shell允许将一组命令集或语句形成一个可用块，这些块称为shell函数。
- 定义函数的两种格式:

```
函数名（）  {                    function 函数名（）  {
   命令  ...                        命令  ...
}                                }
```

- 函数可以放在同一个文件中作为一段代码，也可以放在只包含函数的单独文件中
- 所有函数在使用前必须定义，必须将函数放在脚本开始部分，直至shell解释器首次发现它时，才可以使用

函数（2/3）

- 调用函数的格式为:

  函数名 param1 param2……

- 使用参数同在一般脚本中使用特殊变量

  $1，$2 ...$9一样

- 函数可以使用return 提前结束并带回返回值

  return 从函数中返回，用最后状态命令决定返回值。

  return 0 无错误返回

  return 1 有错误返回

函数（3/3）

例：14_function.sh

```
1  #!/bin/bash
2  is_it_directory()
3  {
4      if [ $# -lt 1 ]; then
5          echo "I need an argument"
6          return 1
7      fi
8      if [ ! -d $1 ]; then
9          return 2
10     else
11         return 3
12     fi
13 }
14 echo -n "enter destination directory:"
15 read direct
16 is_it_directory $direct
17 echo "the result is $?"
```