目录

[1 shell 脚本编程概述 1](#_Toc14070285)

[1.1 shell 概述 1](#_Toc14070286)

[1.2 shell 脚本编程优势 2](#_Toc14070287)

[1.3 第一个shelll脚本例子 2](#_Toc14070288)

[1.3.1 shell 脚本的基本元素 2](#_Toc14070289)

[1.3.2 执行shell脚本 2](#_Toc14070290)

[2 Linux 文件系统和文本编辑器 2](#_Toc14070291)

[2.1 用户和用户管理 2](#_Toc14070292)

[2.1.1 用户管理常用命令 2](#_Toc14070293)

[2.2 文件和目录操作 2](#_Toc14070294)

[2.2.1 文件操作常用命令 2](#_Toc14070295)

[2.2.2 目录操作常用命令 3](#_Toc14070296)

[2.2.3 文件和目录权限管理 3](#_Toc14070297)

[2.2.4 查找文件命令 find 3](#_Toc14070298)

[2.3 文本编辑器 4](#_Toc14070299)

[3正则表达式 5](#_Toc14070300)

[3.1 正则表达式基础 5](#_Toc14070301)

[3.2 正则表达式的扩展 6](#_Toc14070302)

[3.3 通配 6](#_Toc14070303)

[3.4 grep命令 6](#_Toc14070304)

[3.4.1 grep 命令基本用法 6](#_Toc14070305)

[第6章 变量和引用 8](#_Toc14070306)

[6.1 变量 8](#_Toc14070307)

[6.1.1 变量替换和赋值 8](#_Toc14070308)

[6.1.2 无类型的shell脚本变量 9](#_Toc14070309)

[6.1.3 环境变量 10](#_Toc14070310)

[Shell脚本中调用另外一个脚本的方法 11](#_Toc14070311)

# 1 shell 脚本编程概述

## shell 概述

* + 1. Linux 简介
    2. shell 简介
    3. shell的种类和版本

可以通过查看Linux下的相应文件来得知当前Linux中安装的shell种类

# cat /etc/shells

/bin/sh //sh是执行的快捷方式

/bin/dash

/bin/bash

/bin/rbash

bash -version命令查看当前shell的版本号

## 1.2 shell 脚本编程优势

## 1.3 第一个shelll脚本例子

### 1.3.1 shell 脚本的基本元素

whologged.sh:

#！/bin/bash

data #显示日期

who #显示当前的登录用户

#!指定一个文件类型的特殊标记，它告诉linux系统这个文件的执行需要一个解释器。后面

路径指明了解释器在系统的位置。#后面的是注释。

man ls 获取ls命令的帮助信息

### 1.3.2 执行shell脚本

chmod u+x whologged.sh //赋予执行权限

./whologged.sh

# 2 Linux 文件系统和文本编辑器

## 2.1 用户和用户管理

### 2.1.1 用户管理常用命令

用户账号添加命令-----useradd或adduser

useradd [option] [username]

执行该命令的结果是在/etc/passwd文件中增加一行记录，在home目录下创建新用户的主目录。

sudo useradd wang

修改用户账号---usermod

删除用户账号命令----userdel

sudo userdel -r wang

用户口令管理命令----passwd

passwd [option] [username]

sudo passwd wang

sudo tail -l /etc/shadow //查看是否添加成功

## 2.2 文件和目录操作

### 2.2.1 文件操作常用命令

ls 命令

ls -a 显示所有文件，包括隐藏文件

ls -l 显示文件的详细信息

ls -R 递归显示

cp 命令

-a 通常复制目录，保留链接，文件属性，并递归的复制目录

-d 复制时保留链接

-f 删除已经存在的目标文件而不提示

-r 递归复制

-l 不进行复制操作，只是链接文件

mv 命令

rm 命令

### 2.2.2 目录操作常用命令

mkdir命令

-m 对新建目录设置存取权限

-p 可以是一个路径名称。一次可以建立多个目录

mkdir -m 777 tsk

mkdir -p testdir/test

rmdir -p

递归删除目录，删除是目录必须为空

rm -rf 递归的删除目录及目录下文件

cd ~ 返回登录目录

cd - 返回上次访问的目录

pwd 当前目录

chao@ubuntu:~$ cd rk3308\_linux/buildroot/output/

chao@ubuntu:~/rk3308\_linux/buildroot/output$ cd -

/home/chao

### 2.2.3 文件和目录权限管理

chmod命令用于更改文件或目录的访问权限，可以用字母表示法，也可以用数字表示法。

用户类型 数学符号 文件类型

u （user） + 添加某个权限 r 可读

g （group） -取消某个权限 w 可写

o 表示其他用户 = 赋予给定权限并取消其他所有权限 x可执行

a 表示所有用户

数字表示：0没有权限，1可执行权限，2可写权限，4可读权限

chmod u+x,g+x testvim

chmod 764 testvim

chown 命令 更改文件拥有者命令

-R 递归变更拥有者

### 2.2.4 查找文件命令 find

find 路径 选项 操作

.表示当前目录， /表示系统根目录。可以指定按照文件属主，更改时间，文件类型等条件来

查找

name 根据文件名查找文件

perm 根据文件权限查找文件

depth 查找文件时，先查找当前目录中的文件，然后再其子目录中查找

type 查找某一类型的文件

b 块设备文件

d 目录

c 字符设备文件

p 管道文件

l 符号链接文件

f 普通文件

find命令的操作用于指定结果的输出方式

print 将匹配的文件输出到标准输出

exec 对匹配的文件执行该参数所给出的shell命令。相应的命令的形式为’command’ {} \;注意{} \;空格

find . -name ‘t\*’ -perm 744 -print //查找以t开头的，且文件属主具有读写执行权限的文件

find /etc -type f -name “rc\*” -exec ls -l {} \;

## 2.3 文本编辑器

命令模式

vim +n file 编辑file文件并将光标置于第n行

vim + file 编辑file文件，并将光标置于最后一行

保存和退出命令

W 保存文本

Q 退出文本编辑器

q! 退出不保存

Wq 保存并退出

插入模式

在命令模式下按I,o,a等字母都可以进入插入模式。ESC进入命令模式

h 将光标向左移动

j将光标向下移动

k将光标向上移动

l将光标向右移动

} 光标移动到段落结尾

{ 光标移动到段落开头

）光标移动到句子的结尾

（ 光标移动到句子的开头

^ 移动到当前行第一个非空字符

$ 移动到当前行末尾

:n 移动到行n

x 删除光标当前位置字符

dd 删除光标在的整行

d$ 删除当前光标位置到改行结束的所有文件

底行工作模式

/ 进入底行工作模式

输入待搜索字符串后按enter开始搜索

vim 配置

set showmatch:文件中自动显示匹配的括号

set nu：在文件中显示行号

set autoindent: 编辑时自动缩进

set cindent： 按照C语言自动缩进

# 3正则表达式

## 3.1 正则表达式基础

正则表达式完成数据过滤，将不满足正则表达式定义的数据拒绝掉，剩下与正则表达式匹配的数据。

正则表达式包括普通字符和元字符。元字符具有特殊的意义：如\*、，^、[].

基本的正则表达式元字符集合及其意义

\* 0个或多个在\*字符之前的哪个普通字符

. 匹配任意字符

^ 匹配行首，或后面字符的非

$ 匹配行尾

[] 匹配字符集合

\ 转义符，屏蔽一个元字符的特殊意义

\<\> 精确匹配符号

\{n\} 匹配前面字符至少出现n次

\{n,m\} 匹配前面字符出现n~m次

1 、 字符

匹配前面一个普通字符0次或者多次

如hel\*o

\* 符号前面的普通字符是l,\*字符就表示匹配l字符0次或者多次，如helo、hello、hellllo

2、号

点号 .用于匹配任意一个字符，只能匹配一个字符

3、^符号

用于匹配行首，表示行首的字符是^字符后面的那个字符

^cloud

这表示匹配以cloud开头的行。

4、$符号

匹配行尾

micky$ 表示匹配以micky结尾的所有的行。

^$ 匹配空行

5、[]符号

匹配字符集合，该符号支持穷举法列出字符集合的所有元素，也支持使用-符号表示字符集合范围，表面字符集合范围从-左边字符开始，到-右边字符结束。如果要匹配任意一个数字，可以使用3-8所示方法：

匹配任意一个数字

[0123456789]或者[0-9]

[a-z] [A-Z]

^符号表示匹配行首，但是，^符号放到[]符号就不再表示匹配行首了，而是表示取反符号。

6 \符号

转义字符，用于屏蔽一个元字符的特殊意义

7 \<\> 符号

精确匹配符号，该符号利用\符号屏蔽<>

\<the\>该正则表达式精确匹配the这个单词。而不匹配包含the字符的单词。

8 \{\}符号

与\*符号类似，都表示前一个字符的重复。但是\{\}可以至此哪个重复的次数

\{n,\} 匹配前面字符至少出现n次

\{n,m} 匹配前面字符出现n~m次

[a-z] \{5\} 精确匹配5个小写字母

## 3.2 正则表达式的扩展

## 3.3 通配

bash shell本身不支持正则表达式，使用正则表达式的是shell命令和工具，如grep、sed、awk等。shell支持通配.\*符号不再表示其前面字符的重复，而是表示任意位的任意字符。？字符表示一个任意字符，^符号在通配中不代表行首，而是代表取反。

例如：\*.awk匹配所有以.awk结尾的文件。 ls -l \*.awk

以0开头，后面跟1个字符且以.pem为后缀的文件，可以使用0?.pem来匹配

列出以a-h范围内的字母开头,并以.awk结尾的文件，可以用[a-h]\*.awk来匹配这些文件。

{} 符号表示一组表达式的集合。如:{[a-h]\*.awk,0?.pem}

上述通配符表示满足[a-h]\*.awk或0?.pem的所有文件。

## 3.4 grep命令

### 3.4.1 grep 命令基本用法

grep [选项][模式][文件]

在一个或多个文件中搜索满足模式的文本行，模式后的所有字符串被看作文件名

grep 命令的模式十分灵活，可以是字符串，也可以是变量，还可以是正则表达式。只要模式中包含空格，就需要使用双引号将模式引起来。grep支持同时搜索多个文件。

选项：

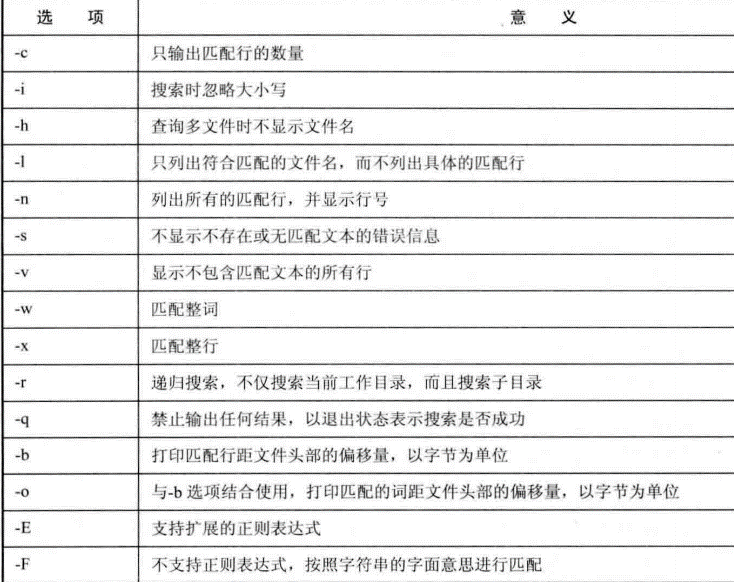
-c :只显示包含模式行的数量，不显示行

-n 列出所有的匹配行并显示行号。

-v 选项不包含模式的所有行

-i 式不区分大小写

-l 选项表示只



# 第6章 变量和引用

## 6.1 变量

变量可以分为本地变量、环境变量和位置参数。

本地变量：仅用在用户当前shell生命周期的脚本中使用的变量

环境变量：使用与所有由登录进程所产生的子进程，就是在用户登录到注销之前的所有程序中都有效。

位置参数：也属于变量，它用于向shell脚本传递参数。

### 6.1.1 变量替换和赋值

变量是某个值的名称，引用变量值就是变量替换。$符号是变量替换符号，如variable是变量名，那么，$ variable就表示变量的值。

变量赋值：

variable=value

${variable=value}

等号的两边可以有空格，如果值中包含空格，则必须用双引号括起来。

variable1 = 33

echo ${variable1}

echo $variable1

variable2=”hello world”

echo $variable2

变量赋值使用另一个变量的值

variable2=”hello world”

variable4=”we are saying $variable2”

echo $variable4

清除变量值

unset variable

其他变量赋值模式

variable=value value值赋给variable,value是一个值不是变量的意思

variable+value 重设变量的值

variable:=value 对未赋值的variable,将value值赋给它

设置只读变量

variable=value

readonly variable

variable.sh

#!/bin/sh

#变量赋值

variable1=33

echo ${variable1}

echo $variable1

variable2="hello world"

echo $variable2

#变量赋值使用另一个变量的值

variable3="we are saying $variable2"

echo $variable3

#清除变量值

unset variable3

echo $variable3

#只读变量

variable4=5

readonly variable4

echo $variable4

#变量赋值模式

echo test....

variable5=12

echo $variable5

colour=black

echo "the background is ${colour:=blue}"

unset colour

echo "the background is ${colour:=blue}"

## 6.1.2 无类型的shell脚本变量

shell脚本变量是无类型的。bash shell不支持浮点型，只支持整型和字符型。如果变量只包含数字shell认定该变量是数值型的，反之，shell认定该变量是字符串。

integer.sh

#!/bin/bash

a=2009

let "a+=1"

echo "a=$a"

b=xx09

echo "b=$b"

#字符型变量默认数值为0,执行强制类型转换不起作用

#declare 强制类型转换

declare -i b

echo "b=$b"

let "b+=1"

echo "b=$b"

exit 0

null-undeclare.sh

#!/bin/bash

c=" "

echo "c=$c"

let "c+=1"

echo "c=$c"

echo "e=$e"

let "e+=1"

echo "e=$e"

exit 0

chao@ubuntu:~/code/shell/chapter6$ ./null-undeclare.sh

c=

c=1

e=

e=1

执行结果可以看出shell变量是无类型的，并且shell变量同时有数值型和字符型两种赋值。数值型初值为0，字符型初值为空。可以不预先定义变量而直接使用它。

### 6.1.3 环境变量

1、定义和清除环境变量

环境变量适用于所有由登录进程所产生的子进程.

ENVIRON-VARIABLE =value #环境变量赋值

export ENVIRON-VARIABLE

用export 命令声明一下，说明此变量为环境变量。环境变量名字一般由大写字母组成。

例子：

APPSPATH=/usr/local

export APPSPATH

echo $APPSPATH

如果要列出系统中所有的环境变量，可以使用env命令。

清除环境变量可以使用unset APPSPATH

注意：使用export生成的环境变量只在当前的控制台有效。

在/etc/profile里面配置的环境变量在shell中使用时，可以先执行source /etc/profile

2、重要的环境变量

PWD和OLDPWD

一个是当前目录，一个是旧的目录。可以通过cd - 回到旧目录。

PATH

用于帮助shell找到用户所输入的命令。PATH记录了一系列的目录列表，shell为每个输入命令搜索PATH中的目录列表。

echo $PATH //显示PATAH环境变量的值

如果需要在PATH中添加新的目录，可以使用下面的命令：

export PATH=”/new directory”:PATH

new directory 就是新加的目录，后面用冒号加$PATH，表示new directory加上旧的PATH

值，得到新的PATH变量值。

HOME

HOME记录当前用户的根目录

SHELL

SHELL变量保存默认的shell，默认值为/bin/bash.

USER和UID

USER表示已登录用户的名字，UID表示登录用户的ID

3、几个环境变量配置文件

.bash\_profile、.bashrc和.bash\_logout

当用户登录时，shell会自动执行.bash\_profile文件。ubuntu12中不存在这个文件，此时登录时会自动执行用户的配置文件/home/用户名/profile文件。

# [Shell脚本中调用另外一个脚本的方法](https://www.cnblogs.com/royfans/p/7761573.html)

 在Linux平台上开发，经常会在console(控制台)上执行另外一个脚本文件，经常用的方法有：**./my.sh** 或 **source my.sh** 或 **. my.sh**；这三种方法有什么不同呢？我们先来了解一下在一个shell脚本中如何调用另外一个shell脚本，其方法有 **fork    exec    source。**

1、fork  ( /directory/script.sh) ：

如果shell中包含执行命令，那么子命令并不影响父级的命令，在子命令执行完后再执行父级命令。子级的环境变量不会影响到父级。

fork是最普通的, 就是直接在脚本里面用/directory/script.sh来调用script.sh这个脚本. 运行的时候开一个sub-shell执行调用的脚本，sub-shell执行的时候,parent-shell还在。

sub-shell执行完毕后返回parent-shell. sub-shell从parent-shell继承环境变量.但是sub-shell中的环境变量不会带回parent-shell

2、exec (exec /directory/script.sh)：

执行子级的命令后，不再执行父级命令。

exec与fork不同，不需要新开一个sub-shell来执行被调用的脚本.  被调用的脚本与父脚本在同一个shell内执行。但是使用exec调用一个新脚本以后, 父脚本中exec行之后的内容就不会再执行了。这是exec和source的区别

3、source (source /directory/script.sh)：

执行子级命令后继续执行父级命令，同时子级设置的环境变量会影响到父级的环境变量。

与fork的区别是不新开一个sub-shell来执行被调用的脚本，而是在同一个shell中执行. 所以被调用的脚本中声明的变量和环境变量, 都可以在主脚本中得到和使用.

以上三种就是调用shell脚本的不同方法，./my.sh即是fork的方法，source my.sh和. my.sh（点加空格加脚本文件）既是source的方法。

在linux系统上，搭建嵌入式开发平台，在交叉编译代码之前，都需要执行脚本设置环境变量，切记需要使用sourc 或 点的方式执行shell脚本，原因如上。

**嵌入式linux无法执行shell脚本问题**

首先查看/etc/shells文件看支持什么类型的shell,一般是bash

bash -version查看板子的shell版本

编写shell脚本例子：

#/bin/bash //非常重要，指明了解释器路径，没有此句将不能执行bash脚本

int=1

while (( int <= 5 ))

do

echo "$int"

let "int++"

done