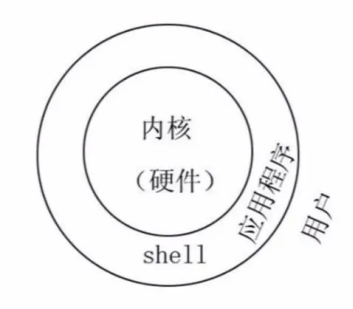
Shell与Bash

原创 小白要努力11 已于 2023-08-14 15:55:20 修改 阅读量309 收藏 4 点赞数 2 版权
文章标签： bash 开发语言

在上篇帖子利用wrf模拟过程中，当运行到./wrf.exe步骤时，发现模式运行速度十分的缓慢，于是下来询问了老师，说是wrf运行速度与域的网格设置以及嵌套层数有关，要提高运行速度可以采用shell脚本的并行运行。又是一个不懂的知识点，于是准备下来学习。

一、什么是shell？shell什么时候启动？shell跟Bash有什么关系

shell是一个**命令语言解释器**，是用户和Linux内核之间的**接口程序**，用户在提示符下输入的每个命令都由shell先解释然后再传给Linux内核（如图）。当用户成功登入系统后shell就启动了，并始终作为你与系统内核的交互手段直至你退出系统。Bash（Bourne Again shell ）是目前**最常用**的一种shell，也是当前大多数Linux发行版的默认Shell。除此之外还有Bourne shell (sh), C shell (csh), 和 Korn shell (ksh)等。



图片来源：https://zhuanlan.zhihu.com/p/56532223

那么，如何查看当前系统中shell的类型呢？

```
$echo $SHELL

/bin/bash
```

echo用于在终端打印出文本。而$是一个新的Shell特殊符号。它提示Shell，后面跟随的不是一般的文本，而是用于存储数据的变量。Shell会根据变量名找到真正的文本，替换到变量所在的位置。SHELL变量存储了当前使用的Shell的信息你可以在bash中用sh命令启动sh，并可以用exit命令从中退出。

二、shell命令

shell命令可以分为以下三类：

- 内建函数(built-in function)：shell自带的功能
- 可执行文件(executable file)：保存在shell之外的脚本，提供了额外的功能。
- 别名(alias)：给某个命令的简称

内建函数是shell自带的预先写好的，实现一定功能的程序。

可执行文件是shell之外的脚本，提供了使用者自定义的功能。Shell必须在系统中找到对应命令名的可执行文件，才能正确执行。我们可以用绝对路径来告诉Shell可执行文件所在的位置。如果用户只是给出了命令名，而没有给出准确的位置，那么Shell必须自行搜索一些特殊的位置，也就是所谓的默认路径。Shell会执行第一个名字和命令名相同的可执行文件。例如，我们可以通过which命令，来确定命令名对应的是哪个可执行文件：

```
$which date
$which pwd
#以上两个命令分别返回date和pwd命令对应的可执行文件的绝对路径
```

别名是给某个命令一个简称，以后在Shell中就可以通过这个简称来调用对应的命令。例如，在Shell中，我们可以用alias来定义别名：

```
$alias freak="free -h"
#即将free -h命令简化成freak
```

可以用alias将一些常用的命令进行简化，比如"ls -l"简化为"ll"。

那么，如何了解命令的类型呢？可以用type命令查看命令类型。

```
$type date
$type pwd
#如果是内建函数会返回builtin字样，如果是可执行文件，将返回文件的路径。
```

三、命令的选项和参数

shell的一行命令中可以包含**选项**和**参数**。总的来说，选项用于控制命令的行为，而参数说明了命令的作用对象。

“-” 叫短选项，用于引领一个英文字母。多个短选项的字母可以合在一起，跟在同一个"-"后面。比如，下面的两个命令就等价：

```
$uname -m -r
$uname -mr          #uname命令用于输出系统信息
```

“--”叫长选项，用于引领一整个英文单词。例如：

```
$date --version
```

一个命令可能有非常多的选项，要弄清楚它们的用法还需要阅读相应的帮助文档。一般情况下是该命令后加“-h”选项，或者是加"--help"选项。

参数说明了命令的作用对象。就拿echo这个命令来说，它能把字符打印到终端。它选择打印的对象，正是它的参数：

```
$echo hello
```

有的时候，选项也会携带变量，以便来说明选项行为的原材料。比如：

```
$sudo date --set="1999-01-01 08:00:00"
```

date是参数，选项"--set"用于设置时间，用等号连接的，就是它的参数。date会把日期设置成这一变量所代表的日期。如果用短选项，那么就要用空格取代等号了：

```
$sudo date -s "1999-01-01 08:00:00"
```

值得注意的是，**Shell对空格敏感。当一整个参数信息中包含了空格时，我们需要用引号把参数包裹起来，以便Shell能识别出这是一个整体**。

四、shell中的特殊符号

1）注释符

#，除了shell脚本第一行“#!/bin/bash”里的#特殊

2）美元符

$，变量符。与反斜杠转义符相反，使其后面的普通字符作为变量名，如$a表示变量a的值。变量字符长度超过1个时，用{}括起来，如$abc。

3）单引号

'，用单引号引起来的字符全部做普通字符，即全部原样。如:

```
$echo ‘my $SHELL’

‘my /bin/bash’
```

4）双引号

"，引号内的内容，除$、转义符\、倒引号`这三个保留特殊功能，其他字符均做普通字符。

5）倒引号（键盘数字1左边那个键）

`，引号内的字符串当做shell命令行解释执行，得到的结果取代整个倒引号括起来的部分。

```
$LOGNAME=quanrui
$echo "我当前目录是`pwd`，我的登录姓名是$LOGNAME"

我当前目录是/home/quanrui，我的登录姓名是quanrui
-----------------------------------------------------
$echo 'my home is $HOME'

my home is $HOME
-----------------------------------------------------
$echo "my home is $HOME"

my home is /root
-----------------------------------------------------
$echo `my home is $HOME`
```

6）反斜线

\，反斜线是转义字符，即能把特殊字符变成普通字符。在某个字符前面利用反斜杠（\）能够阻止shell把后面的字符解释为特殊字符。

```
$echo "Filename is N0\$\*"

Filename  is  N0$*
```

注意：在单引号括起来的字符串中，反斜线也成为普通字符，而失去转义字符功能。

五、变量

shell变量可以保存如路径名、文件名、数字等。

1）变量赋值

shell利用“变量名=值”来表示赋值，等号两边不能有空格。比如：

```
$var=World
```

如果取值包含空格，必须用单引号或双引号括起来，例如：

```
$var='abc bcd'
$var="abc bcd"
```

此外，在Bash中还可以把一个命令输出的文本直接赋予给一个变量：

```
$now=`date`
#借助``符号，date命令的输出存入了变量now。
```

还可以把一个变量中的数据赋值给另一个变量：

```
$echo "${var} is good!"

Word is good
```

注意：Shell变量可以同时用大小写字母，shell区分大小写

2）引用变量

shell可以用变量前加$的方式来引用变量，所谓的引用变量就是把变量翻译成变量中存储的文本。比如：

```
$var=Word
$echo $var

Word
----------------------
$echo "Hello $var"

Hello Word
```

为了避免变量名和尾随的普通文本混淆，也可以用换用${}的方式来标识变量。比如：

```
$echo "${var} is good!"

World is good!
```

3）变量分类

shell变量可分为以下三类：

- **局部（用户/本地）变量：**局部变量只在创建它们的Shell中使用，可以在shell程序内任意使用和修改它们。
- **环境变量：**可以在创建它们的Shell及其派生出来的任意子程序中使用。有些变量是用户创建的，其他的则是专用的（比如PATH、HOME)。是系统环境的一部分，不必去定义它们，可以在shell程序中使用它们 。还能在shell中加以修改。
- **内部变量 ：**由系统提供，与环境变量不同，用户不能修改它们。

1.局部变量

a.显示变量

echo命令可以显示单个变量取值，变量名前加$

b.清除变量

unset 变量名，如：

```
name=welcome
$echo ${name}

welcome
-----------------
$unset name
$echo ${name}

                    #空
```

c.设置只读变量

设置变量时，不想再改变其值，可以将其设为只读变量。即：

变量名=值

readonly 变量名

```
$free=book
$echo $free

book

$readonly free
$free=www
bash:free:readonly variable
```

2.环境变量

环境变量用于所有用户进程（通常称为子进程）。登陆进程称为父进程，通过pstree可以查看。

- 环境变量可以用于所有子程序，着包括编辑器、脚本和应用
- 环境变量可以在命令行中设置，但用户注销时这些值将丢失
- 环境变量均为大写
- 必须用export命令导出

```
#设置环境变量

$VARIABLE-NAME=value
$EXPORT VARIABLE-NAME

------------------------
#显示环境变量

env                    #可以看到所有的环境变量
echo $环境变量名        #显示一个变量

------------------------
#清除环境变量

unset 环境变量名
```

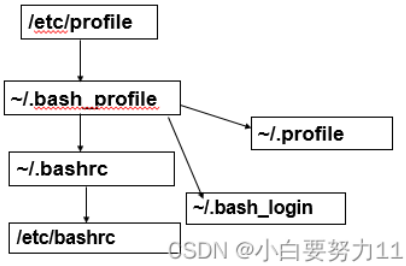
- HOME： 代表使用者的根目录。cd ~ 去到使用者的根目录 或者利用 cd 就可以直接回到使用者家目录了。
- PS1：shell的主提示符
- SHELL： 目前这个环境使用的 SHELL 是哪个程序？ 如果是 bash 的话，预设是 /bin/bash
- PWD：用户当前工作目录的路径。它指出用户目前在Linux文件系统中处在什么位置。它是由Linux自动设置的
- HISTSIZE： 这个与“历史命令”有关，曾经下达过的指令可以被系统记录下来，而记录的“数目”则是由这个值来设定的。
- ENV： 这个使用者所使用的个性化环境设置文件的读取文件。
- MAIL： 当我们使用 mail 这个指令在收信时，系统会去读取的邮件信箱文件 （mailbox)。
- PATH： 就是执行文件搜寻的路径，目录与目录中间以冒号(:)分隔， 由于文件的搜寻是依序由 PATH 的变量内的目录来查询，所以，目录的顺序也是重要的喔。
- LANG： 语系文件，很多数据都会用到他，当出现编码错误的时候往往需要设置它，中文编码是zh_CN.UTF8

环境变量设置完成后，利用以下命令让配置起作用

```
$source 配置文件名
```

让环境变量的修改在退出shell再次登录时仍有效，需要在相关配置文件中修改。

Bash的初始化文件有：/etc/profile、~/.bash_profile、~/.bash_login、~/.profile、~/.bashrc、/etc/bashrc



- /etc/profile 存放一些全局（共有）变量，不管哪个用户，登录时都会读取该文件。通常设置一些Shell变量PATH,USER,HOSTNAME和HISTSIZE等
- ~/.bash_profile:每个用户都可使用该文件输入专用于自己使用的shell信息,当用户登录时,该文件仅仅执行一次!默认情况下,他设置一些环境变量,执行用户的.bashrc文件.
- ~/.bashrc:该文件包含专用于你的bash shell的bash信息,当登录时以及每次打开新的shell时,该该文件被读取.
- /etc/bashrc:为每一个运行bash shell的用户执行此文件.当bash shell被打开时,该文件被读取.

  上诉配置文件的作用
  1）登录Linux先启动系统配置文件/etc/profile，并从/etc/profile.d目录的配置文件中搜集shell的设置，为系统的每个用户设置环境信息。
  2）用户配置文件~/.bash_profile，每个用户专用于自己使用的shell信息,仅用户登录时执行一次!
  默认情况下,此文件通过脚本执行同目录下用户的.bashrc文件。
  3）~/.bashrc文件包含专用于用户bash shell的bash信息,登录及每次打开新的shell时都会执行。里面又会调用/etc/bashrc

3.内部变量

部分内部变量：

- $# ——传送给shell程序的位置参数的数量
- $?——最后命令的完成码或者在shell程序内部执行的shell程序（返回值）
- $0——当前shell程序的名称
- $*——调用shell程序时所传送的全部参数的单字符串， “参数1”“参数2”…形式保存的参数
- $@ “参数1”“参数2”…形式保存的参数
- $n 第n个参数
- $$ 本程序的PID
- $! 上一个命令的PID

4.位置参数

a.位置参数及引用

一个shell脚本，当从命令行或者从其他shell脚本中调用它的时候，这个脚本接收若干参数。这些选项是通过Linux作为位置参数（positional parameter）提供给shell程序的。在shell脚本中应有变量，接收实参，这类变量的名称很特别，分别是1，2，3，…，这类变量称为位置变量。位置参数1存放在位置变量1中，位置参数2存放位置变量2中，……，在程序中可以使用$1，$2，……来访问。例如：

```
$vi test_arg.bash:

#!/bin/bash

echo $0          #当前shell程序的名称
echo $1          #位置变量1接收的第一个实参
echo $2          #位置变量2接收

$./rest_arg.bash hello word


test_arg.bash
hello
word
```

b.用shell命令为位置参数赋值

在shell程序中可以利用set命令为位置参数赋值或重新赋值。

一般格式：set [参数表]

说明：该命令后面无参数时，将显示系统中的系统变量的值；如果有参数将分别给位置参数赋值。

六、变量表达式

1）条件判断命令test

test n1 -参数 n2（或 –参数 表达式），真返回0，假返回1。

a.整数

参数列表：

-lt 小于

-le 小于等于

-gt 大于

-ge 大于等于

-eq 等于

-ne 不等于

```
$test 1 -lt 4        #判断1<4
echo $?              #结果为0
```

上述语句还存在等价形式：[1 -lt 4]，即用中括号代替test，把表达式括起来实现判断，注意表达式与中括号间有空格。

b.文本判断

文本相同: $test abc = abx; echo $?

文本不同： $test abc != abx; echo $?

按照词典顺序，一个文本在另一个文本之前： $test apple > tea; echo $?

按照词典顺序，一个文本在另一个文本之后： $test apple < tea; echo $?

c.文件测试

参数列表：

-e 文件是否存在

-f 文件是否存在且是普通文件

-d 文件是否存在且是目录文件

-s 文件是否存在且字节数大于0

-r 文件是否存在且可读

-w 文件是否存在且可写

-x 文件是否存在且可执行

-L 文件是否存在且是软连接

如：test -d “mydoc”，就是判断mydoc是否是目录。

d.字符串测试

列表：

test s 字符串s非空

test s1=s2 字符串s1等于s2

test s1!=s2 字符串s1不等于s2

test -z s 字符串长=0,即为空串

test -n 字符串长>0

e.其他参数

-a 逻辑与

```
expression1 –a expression2
```

-o 逻辑或

```
expression1 –o expression2
```

! 逻辑非

```
! expression
```

七、数学运算

在Bash中，数字和运算符都被当做普通文本，比如：

```
$result=1+2
$echo $result

#Bash不会进行任何运算。它只会打印文本“1+2”。
```

所以在Bash里，用户需要通过$((...))语法来进行数值运算，在双括号中可以放入整数的加减乘除表达式，Bash则会对其中的内容进行数值运算。比如：

```
$echo $((2 + (5*2)))

12
```

同时，在$((...))中，也可以使用变量。比如：

```
$var=1
$echo $(($var + (5**2)))

26
```

八、返回代码（详细参考 快速学习Bash 第6部分）

在Shell中，用户运行了程序后，可以通过$?变量来获知返回码。

```
$gcc foo.c
$./a.out
$echo $?
```

当程序正常运行时，程序返回代码一般是0。如果一个程序运行异常，那么这个程序将返回非0的返回代码。

九、Bash脚本

1）shell脚本的一般结构

a.#! /bin/bash （必须指出shell类型）

b.#注释 （解释脚本的功能）

c.函数

d.主过程

例如一个shell脚本：

```
$ vi clearup.sh

#!/bin/bash
# this is clear
cd /var/log
cat /dev/null>/var/log/messages        #/dev/null是Linux中的无限大的垃圾回收站
echo "Logs are cleared up."
```

```
$vi abc.sh                          #创建名为abc.sh的脚本

#!/bin/bash                        #指出shell类型
#a simple shell script example
#a function
function sayhello()                #函数
{
echo "Enter Your name:"
read name
echo "Hello $name"
}
echo "Programe starts here..."     #主过程
sayhello
echo "Programe ends."
```

2)shell脚本运行的一般步骤：

a.编辑文件

b.保存文件

c.将文件赋予可执行的权限

d.运行及排错

3）脚本的返回代码

用户可在脚本的末尾用exit命令来设置脚本的返回代码，例如：

```
$vi hello_world.bash

#!/bin/bash
echo Hello
echo World
exit 0                  #如果脚本正常运行，则返回0
```

此外，用户可通过$?变量查询脚本的返回代码

```
$./hello_world.bash
$echo $?
```

如果在脚本中部出现exit命令，脚本会直接在这一行停止，并返回该exit命令给出的返回代码。

```
$vi demo_exit.bash

echo hello
exit 1 echo world
```

用户可以运行该脚本，检查其输出结果，并查看其返回代码。

十、函数

在定义函数时，可以用花括号{}来标识函数包括的部分：

```
#!/bin/bash

function my_info ()                #定义函数my_info是函数名，function可要可不要
{
  lscpu >> log
  uname –a >> log
  free –h >> log
}

my_info                            #调用函数，进行执行，调用时只需要一个函数名就行
```

像脚本一样，函数调用时还可以携带参数。在函数内部，用户可以用$1、$2这种形式的变量来使用参数：

```
#!/bin/bash

function my_info (){
  lscpu >> $1
  uname –a >> $1
```

```
  free -h >> $1 7 }

my_info output.file                  #调用时携带参数utput.file
my_info another_output.file          #调用时携带参数another_output.file
```

十一、跨脚本调用

在Bash中使用source命令，可以实现函数的跨脚本调用。命令source的作用是在同一个进程中执行另一个文件中的Bash脚本。例如，有两个脚本my_info.bash和app.bash。脚本my_info.sh中的内容是：

```
#!/bin/bash

function my_info (){
  lscpu >> $1
  uname -a >> $1
  free -h >> $1
}
```

脚本app.bash中的内容是：

```
#!/bin/bash

source my_info.bash
my_info output.file
```

运行app.bash时，执行到source命令那一行时，就会执行my_info.bash脚本。在app.bash的后续部分，就可以使用my_info.bash中的my_info函数。

十二、逻辑判断

Bash除了可以进行数值运算，还可以进行逻辑判断。其中存在选择和循环两种语法结构，这两种语法结构可以改变脚本的运行顺序，从而编写出更加灵活的程序。

1）选择结构

选择结构是一种语法结构，可以让程序根据条件决定执行哪一部分的指令。最早的程序都是按照指令顺序依次执行。选择结构打破了这一顺序，给程序带来更高的灵活性。例如：

```
$vi demo_it.bash

#!/bin/bash

var=`whoami`

if [ $var = "root" ]
then
  echo "You are root"
  echo "You are my God."
fi
```

这个脚本中使用了最简单的if结构。关键字if后面跟着中括号[]，里面是一个逻辑表达式。这个逻辑表达式就是if结构的条件。如果条件成立，那么if将执行then到fi之间包含的语句，我们称之为隶属于then的代码块。如果条件不成立，那么then的代码块不执行。这个例子的条件是判断用户是否为root。因此，如果是非root用户执行该脚本，那么Shell不会打印任何内容。

还可以通过if...then...else...结构，让Bash脚本从两个代码块中选择一个执行。该选择结构同样有一个条件。如果条件成立，那么将执行then附属的代码块，否则执行else附属的代码块。下面的demo_if_else.bash脚本是一个小例子：

```
#!/bin/bash

filename=$1

if [ -e $filename ]
then
  echo "$filename exists"
else
  echo "$filename NOT exists"
fi

echo "The End"
```

if后面的“-e $filename”作为判断条件。如果条件成立，即文件存在，那么执行then部分的代码块。如果文件不存在，那么脚本将执行else语句中的echo命令。末尾的fi结束整个语法结构。脚本继续以顺序的方式执行剩余内容。运行脚本：

```
$./demo_if_else.bash a.out
```

脚本会根据a.out是否存在，打印出不同的内容。

我们看到，在使用if...then...else...结构时，我们可以实现两部分代码块的选择执行。而在then代码块和else代码块内部，我们可以继续嵌套选择结构，从而实现更多个代码块的选择执行。比如脚本demo_nest.bash:

```
#!/bin/bash

var=`whoami`
echo "You are $var"

if [ $var = "root" ]
then
   echo "You are my God."
else
  if [ $var = "vamei" ]
  then
     echo "You are a happy user."
   else
     echo "You are the Others."
   fi
fi
```

在Bash下，我们还可以用case语法来实现多程序块的选择执行。比如下面的脚本demo_case.bash：

```
#!/bin/bash

var=`whoami`
echo "You are $var"
case $var in
root)
  echo "You are God."
;;
vamei)
  echo "You are a happy user."
;;
*)
  echo "You are the Others."
;;
esac
```

这个脚本和上面的demo_nest.bash功能完全相同。可以看到case结构与if结构的区别。关键字case后面不再是逻辑表达式，而是一个作为条件的文本。后面的代码块分为三个部分，都以文本标签)的形式开始，以;;结束。在case结构运行时，会逐个检查文本标签。当条件文本和文本标签可以对应上时，Bash就会执行隶属于该文本标签的代码块。

文本标签除了是一串具体的文本，还可以包含文本通配符。结构case中常用的通配符包括：

```
通配符     含义              文本标签例子          符合条件的文本
*       任意文本                 *)              Xyz, 12a3, …
?       任意一个字符            a?c)              abc, axc, …
[]    范围内一个字符          [1-5][b-d])          2b, 3d, …
```

上面的程序中最后一个文本标签是通配符*，即表示任意条件文本都可以触发此段代码块的运行。当然，前提是前面的几个文本标签都没有“截胡”。

2）循环结构

循环结构是编程语言中另一种常见的语法结构。循环结构的功能是重复执行某一段代码，直到计算机的状态符合某一条件。在**while语法**中，Bash会循环执行隶属于while的代码块，直到逻辑表达式不成立。比如下面的demo_while.bash：

```
#!/bin/bash

now=`date +'%Y%m%d%H%M'`
deadline=`date --date='1 hour' +'%Y%m%d%H%M'`

while [ $now -lt $deadline ]
do
   date
   echo "not yet"
   sleep 10
   now=`date +'%Y%m%d%H%M'`
done
```

```
13 echo "now, deadline reached"
```

关键字do和done之间的代码是隶属于该循环结构的代码块。在while后面跟着条件，该条件决定了代码块是否重复执行下去。这个条件是用当前的时间与目标时间对比。如果当前时间小于目标时间，那么代码块就会重复执行下去。否则，Bash将跳出循环，继续执行后面的语句。

如果while的条件始终是真，那么循环会一直进行下去。下面的程序就是以无限循环的形式，不断播报时间：

```
#!/bin/bash

while true
do
  date
  sleep 1
done
```

语法while的终止条件是一个逻辑判断。如果在循环过程中改变逻辑判断的内容，那么我们很难在程序执行之前预判循环进行的次数。正如我们之前在demo_while.bash中看到的，我们在循环进行过程中改变着作为条件的逻辑表达式，不断地更新参与逻辑判断的当前时间。与while语法对应的是**for循环**。这种语法会在程序进行前确定好循环进行的次数，比如demo_for.bash：

```
#!/bin/bash

for var in `ls log*`
do
  rm $var
done
```

在这个例子中，命令ls log*将返回所有以log开头的文件名。这些文件名之间由空格分隔。循环进行时，Bash会依次取出一个文件名，赋值给变量var，并执行do和done之间隶属于for结构的程序块。由于ls命令返回的内容在是确定的，因此for循环进行的次数也会在一开始确定下来。

十三、命令结果重定向

1　　stdout 标准输出

2　　stderr 标准错误

编写shell脚本时，常用

```
command >file 2>&1
```

该语句的用法为：将stdout直接送向file, stderr继承1的管道后,再被送往file,此时,file 只被打开了一次,也只使用了一个管道FD1,它包括了stdout和stderr的内容。

参考博文：

超算入门笔记：大型机上如何运行WRF模式？一文总结（并行运算、Linux基础、作业调度、WRF运行

如何在 Linux 中创建并运行 Shell 脚本（Bash 初学者教程）

Bash编程入门-1：Shell与Bash

快速学习Bash - 知乎 (zhihu.com)

shell/bash脚本命令教程_bash脚本教程 12-9

shell/bash其实就是我们日常在unix系统终端中执行的语句,只是通常我们在命令行中都是单行语句执行的,而有时,我们希望将一些操作命令写到一个文本中,让电脑自动按顺序或是…

bash shell学习笔记 最新发布 09-22

使用LINUX命编写脚本。bash快捷键、Linux有关网络配置的命令 一、创建shell脚本、重定向输入与输出、执行数学运算、退出脚本 二、shell脚本中的各种结构化命令的格式与用法…

Linux：Shell和Bash 燕双嘤 441

Shell是操作系统与用户进行交互操作的界面. Shell是内核kernel的"壳", 是用来将机器语言和人类语言相互转化而存在的软件层次. Shell是命令语言、命令解释程序及程序设计语言的…

Shell与Bash学习 12-4

1.1.查看Linux下的shell zhangsan@Aliyun:~$cat/etc/shells 1 该命令可以查看我们当前在Linux里安装的shell内容。 1.2.判断bash,shell的内置命令 zhangsan@Aliyun:~$type[-tpa]nam…

Linux Shell概述和Bash基本功能_shell和bash的功能 11-23

Bash: Bash与sh兼容,是Linux系统默认使用的Shell。 其实可以这样理解:Shell是一种编程语言,而Bash是它的语法。 1.3 查看你的Linux所支持的Shell [cfp@bogon ~]$cat/etc/shells …

Shell与Bash介绍 qq_28284627的博客 2750

Linux图形化桌面算不上精美。幸好，Linux提供了更好的与树莓派互动的方式：Shell。 1、Shell和Bash：hell是运行在终端中的文本互动程序；bash（GNU Bourne-Again Shell）是…

Linux基础篇（一）-- Shell与Bash的区别和联系 CarpeDiem 6263

本文主要对学习的Shell和Bash相关知识点进行了总结，对于每一部分的详细介绍，可以参考《鸟哥的Linux私房菜：基础学习篇》和文末推荐的博客进行深入了解。全文主要介绍了S…

Linux下shell脚本：bash的介绍和使用（详细） weixin_42432281的博客 5万+

Shell：一般我们是用图形界面和命令去控制计算机，真正能够控制计算机硬件（CPU、内存、显示器等）的只有操作系统内核（Kernel），由于安全、复杂、繁琐等原因，用户不能…

Shell(bash) 介绍 liaowenxiong的博客 1103

文章目录Shell 介绍Shell 种类命令行环境终端模拟器命令行提示符进入和退出方法Shell 脚本第一个shell脚本BashBash如何解析命令Shell 和 Bash 的历史 Shell 介绍 简单点理解，就…

命令二：Shell常用命令 chang_jinling的专栏 284

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。 Shell 既是一种命令语言，又是一种程序设计语言。 echo 命令用于向窗口输出文本 Shell 的 echo 指令与 PHP 的 e…

Bash和shell的关系 qq_41156733的博客 1579

(#!/bin/bash) 和 (#!/usr/bin/env bash)

linux基础：shell中的sh与bash区别与常用命令 鄒忐的博客 1万+

目录 一，什么是shell和bash? 二，shell的基本用法 三，bash的基本用法 一，什么是sh和bash? #!/bin/sh是#!/bin/bash的缩减版 Linux系统中的/bin/sh本是bash的符号链接，鉴于bas…

什么是Bash、什么是shell? 程序猿进化梯 914

什么是Shell? shell是用户和Linux（或者更准确的说，是用户和Linux内核）之间的接口程序。你在提示符下输入的每个命令都由shell先解释然后传给Linux内核。 shell 是一个命令语…

C Shell 和 Bash的区别 田辛_DensinTian的专栏 9509

在公司写C Shell习惯了，在家里写Bash还真有点不习惯。下面将发现

bash手册 程序人生 2013

bash手册 蓝森林 http://www.lslnet.com 2000年3月28日 22:53作 者： con前言 本文译自《Slackware Linux Unleashed》（第三版）一书的 bash 一章，但做了一些必要的删节，…

bash和shell的区别 MonMama的博客 2万+

Linux 中的 shell 有很多类型，其中最常用的几种是: Bourne shell (sh)、C shell (csh) 和 Korn shell (ksh), 各有优缺点。Bourne shell 是 UNIX 最初使用的 shell，并且在每种 UNIX 上…

shell脚本第一行：#!/bin/bash的含义 热门推荐 花花世界 7万+

相信有接触过shell脚本的同学们都应该知道，shell脚本的第一行一般会写有以下字样： #!/bin/bash或者 #!/bin/sh或者 #!/bin/awk 比较常见的说法是：第一行的内容指定了shell脚本解…

bash shell 中$(),${}, $[] ,$(()), [ ] , [[ ]], (()) ,``的区别和作用 sdc20102010的博客 7705

在bash shell 中 一下符号代表不同的意义： $()和`` 是一组他在shell 中表示的是 命令的替换，就是在shell 中fork 一个子进程 区做 他们 括起来的命令 然后在返回父进程。 特别要注…

#!/bin/sh & #!/bin/bash 周自信的技术博客 4880

在shell脚本的开头往往有一句话来定义使用哪种sh解释器来解释脚本。 目前研发送测的shell脚本中主要有以下两种方式： (1) #!/bin/sh (2) #!/bin/bash 在这里求教同福客栈的各位大…

bash与shell 07-27

bash是shell的一种版本，是当前大多数Linux发行版的默认Shell。而shell是一个总称，指的是运行在终端中的文本互动程序。

1

Bourne Again shell (bash)是对Bourne shell的扩展，完全向后兼容，并在其基础上增加和增强了很多特性。bash也包含了很多C和Korn shell的优点，具有灵活和强大的编程接口，同…
又有友好的用户界面。

2

bash的出现是为了解决Bourne shell在处理用户输入方面的不便，使命令的输入更加容易。

2

所以，bash可以看作是shell的一种更加强大和方便的版本。

1

引用[.reference_title] - *1* [Shell与Bash介绍](https://blog.csdn.net/qq_28284627/article/details/124509159)[target="_blank" data-report-click={"spm":"1018.2226.3001.9630","…

“相关推荐”对你有帮助么？

非常没帮助 没帮助 一般 有帮助 非常有帮助

小白要努力11
码龄1年 暂无认证

2	126万+	12万+	618	
原创	周排名	总排名	访问	等级

66	1	2	3	4
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

Shell与Bash 306

linux系统下安装运行ncview 171

最新评论

linux系统下安装运行ncview
CSDN-Ada助手: 非常感谢您的第二篇博客！您对如何在Linux系统下安装运行ncv...

Shell与Bash
CSDN-Ada助手: 这篇博客对于Shell与Bash的介绍非常详细，读了之后对这两个概念...

您愿意向朋友推荐“博客详情页”吗?

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

linux系统下安装运行ncview

2023年 2篇

目录