

# Linux Shell编程及自动化运维实现——数组和函数

原创

一荤两素

🕒 于 2021-04-21 19:19:12 发布

👁 86

🌟 收藏

版权

分类专栏：


shell

Linux

 文章标签：

shell

运维

 shell

同时被 2 个专栏收录 ▾

0 订阅

4 篇文章

订阅专栏

## Linux Shell编程及自动化运维实现 第4章 数组和函数

### 一、数组

简介

- 1

变量：用一个固定的字符串，代替一个不固定字符串。
- 2

数组：用一个固定的字符串，代替多个不固定字符串。

类型

- 1

普通数组：只能使用整数作为数组索引
- 2

关联数组：可以使用字符串作为数组索引

#### 1、普通数组

##### 1、定义普通数组

方法一：

```
1 # array2=(tom jack alice)
2 # array3=(`cat /etc/passwd`) 希望是将该文件中的每一个行作为一个元数赋值给数组array3
3 # array4=(`ls /var/ftp/Shell/for*`)
4 # array5=(tom jack alice "bash shell")
5 # colors=($red $blue $green $recolor)
6 # array6=(1 2 3 4 5 6 7 "linux shell" [20]=saltstack)
```

方法二：

```
1 数组名[下标]=变量值
2  # array1[0]=pear
3  # array1[1]=apple
4  # array1[2]=orange
5  # array1[3]=peach
6  查看数组
7  [root@localhost ~]# declare -a | grep array1
8  declare -a array1='([0]="pear" [1]="apache" [2]="orange" [3]="peach") '
9  查看数组
10 [root@localhost ~]# echo ${array1[@]}
11 pear apache orange peach
```

2、访问普通数组元素

```
1 访问数组元数：
2  # echo ${array1[0]} 访问数组中的第一个元数
3  # echo ${array1[@]} 访问数组中所有元数 等同于 echo ${array1[*]}
4  # echo ${#array1[@]} 统计数组元素的个数
5  # echo ${!array2[@]} 获取数组元素的索引
6  # echo ${array1[@]:1} 从数组下标1开始
7  # echo ${array1[@]:1:2} 从数组下标1开始，访问两个元素
```

2、关联数组

1、定义关联数组

切记：先声明关联数组

```
1  # declare -A  ass30
```

```
1  方法一
2  数组名[索引]=变量值
3  # declare -A  ass_array1
4  # ass_array1[index1]=pear
5  # ass_array1[index2]=apple
6  # ass_array1[index3]=orange
7  # ass_array1[index4]=peach
8  查看
9  [root@localhost ~]# echo ${ass_array1[@]}
10 peach pear apple orange
```

修改某个值

```
1 [root@localhost-111 ~]# unset lol
2 [root@localhost-111 ~]# declare -A lol
3 [root@localhost-111 ~]# lol[down1]=aaa
4 [root@localhost-111 ~]# lol[down2]=bbb
5 [root@localhost-111 ~]# lol[down3]=ccc
6 [root@localhost-111 ~]# echo ${lol[@]}
7 aaa bbb ccc
8 [root@localhost-111 ~]# lol[down2]=ddd
9 [root@localhost-111 ~]# echo ${lol[@]}
10 aaa ddd ccc
```

方法二

```
1 一次赋多个值
2 # declare -A ass_array2
3 # ass_array2=([index1]=tom [index2]=jack [index3]=alice [index4]='bash shell')
```

2、查看数组

```
1 # declare -A
2 declare -A ass_array1='([index4]="peach" [index1]="pear" [index2]="apple" [index3]="orange" )'
3 declare -A ass_array2='([index4]="bash shell" [index1]="tom" [index2]="jack" [index3]="alice" )'
```

3、访问数组元素

```
1 # echo ${ass_array2[index2]} 访问数组中的第二个元数
2 # echo ${ass_array2[@]} 访问数组中所有元数 等同于 echo ${array1[*]}
3 # echo ${#ass_array2[@]} 获得数组元数的个数
4 # echo ${!ass_array2[@]} 获得数组元数的索引
```

3、数组和循环

1、通过循环定义和显示数组

2、通过数组统计数据

案例1：while脚本快速定义数组

1、定义数组

```
1 #!/bin/bash
2 #循环读取文件，定义数组
```

```
3 while read line
4 do
5 #hosts:数组名
6 #[++i]:索引递增,++i是1开始, i++是0开始。
7 #line:值, 即文件中的内容
8     hosts[++i]=$line
9 done < /etc/hosts
10
11
12
13 #输出索引每一行
14 for i in ${!hosts[@]}
15 do
16 echo "$i :   ${hosts[$i]}"
17 done
```

## 2、测试数组

```
1 [root@localhost ~]# bash array1.sh
2 “数组hosts first:127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4”
3
4 1 :   127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
5 2 :   ::1        localhost localhost.localdomain localhost6 localhost6.localdomain6
6
```

## 案例2：for脚本快速定义数组

### 1、定义数组

```
1 [root@localhost ~]# vim for_array.sh
2 #!/bin/bash
3 #2020
4 #for array
5 for line in `cat /etc/hosts`
6 do
7     hosts[++i]=$line
8 done
9 for i in ${!hosts[@]}
10 do
11 echo "$i:   ${hosts[$i]}"
12 done
13
```

2、测试数组

```
1 [root@localhost ~]# bash for_array.sh
2 1: 127.0.0.1
3 2: localhost
4 3: localhost.localdomain
5 4: localhost4
6 5: localhost4.localdomain4
7 6: ::1
8 7: localhost
9 8: localhost.localdomain
10 9: localhost6
11 10: localhost6.localdomain6
12
```

案例3：数组统计性别

1、定义性别文件

```
1 [root@localhost ~]# vim sex.txt
2 jack m
3 alice f
4 tom m
```

2、定义脚本统计性别

```
1 #!/bin/bash
2 declare -A sex
3 while read line
4 do
5     type=`echo $line|awk '{print $2}'`
6     let sex[$type]++
7 done < sex.txt
8
9 for i in ${!sex[@]}
10 do
11     echo "$i : ${sex[$i]}"
12 done
```

3、测试脚本

```
1 | [root@localhost ~]# bash sex.sh
2 | f : 1
3 | m : 2
```

案例4：使用数组统计，用户shell的类型和数量

1、示例

```
1 | declare -A shells
2 | while read ll
3 | do
4 | type=`echo $ll | awk -F:  '{print $NF}'`
5 | let shells[$type]++
6 | done  < /etc/passwd
7 |
8 | for i in ${!shells[*]}
9 | do
10 | echo "$i: ${shells[$i]}"
11 | done
```

2、验证结果

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-L9L8eQZC-1619003860297)(C:\Users\袁威\AppData\Roaming\Typora\typora-user-images\image-20210421102522753.png)]

二、函数

概念

```
1 | 函数是一段完成特定功能的代码片段 （块）
2 | 在shell中定义了函数，就可以使代码模块化，便于复用代码
3 | 注意函数必须先定义才可以使用。
```

重点

```
1 | 传参 $1,$2
2 | 局部变量 local
3 | 返回值 return    即$?
```

1、定义函数

方法一

```
1 | 函数名() {
2 | 函数要实现的功能代码
3 | }
```

方法二

```
1 | 方法二：
2 | function 函数名 {
3 | 函数要实现的功能代码
4 | }
```

2、调用函数

语法

```
1 | 函数名
2 | 函数名 参数1 参数2
```

三、示例

示例1：初识函数

通过shell脚本，编写系统工具箱

编写循环脚本，功能菜单

provide these tools:

show disk info(d)

show mem info(m)

show cpu info©

quit(q)

```
1 | #!/bin/bash
2 | show_menu() {
3 | cat << EOF
4 | provide these tools:
5 |   show disk info(d)
6 |   show mem info(m)
7 |   show cpu info(c)
8 |   quit(q)
9 | EOF
10 | }
11 |
```

```
12 while :
13 do
14
15 show_menu
16 read -p "Input choice: " choice
17     case $choice in
18         d)
19             echo "=====disk info======"
20             df -hT
21             ;;
22         m)
23             echo "=====meme info======"
24             free -m
25             ;;
26         c)
27             echo "=====cpu info======"
28             uptime
29             ;;
30         q)
31             break
32             ;;
33         *)
34             show_menu
35             ;;
36     esac
37 done
```

## 示例2：阶乘函数（传参）

思路：1 了解阶乘概念——>2 定义函数——>3 引用函数

```
1  #!/bin/bash
2  #定义函数名fun1
3  fun1() {
4  #定义阶乘元数
5  factorial=1
6  #使阶乘循环
7  for((i=1;i<=5;i++))
8  do
9  #阶乘公式
10 factorial=${factorial*$i}
11 done
12 #输出阶乘结果
13 echo "5的阶乘是:$factorial"
```



```
14 }
15 fun1
```

优化1：传参，让函数能够自定义 (i<=\$1 参数1是阶乘的上限)

```
1  #!/bin/bash
2  fun1() {
3    factorial=1
4    for((i=1;i<=$1;i++))
5    do
6      factorial=$((factorial*i))
7    done
8    echo "$1的阶乘是:$factorial"
9  }
10
11  fun1 3
12  测试脚本
```

优化2：传参2，由脚本外部传递参数。

```
1  #!/bin/bash
2  fun1() {
3    factorial=1
4    for((i=1;i<=$1;i++))
5    do
6      factorial=$((factorial*i))
7    done
8    echo "$1的阶乘是:$factorial"
9  }
10
11  fun1 $1
12  fun1 $2
13  fun1 $3
14
15  测试：
16  [root@localhost ~]# bash cc.sh 3 5 10
17  "3的阶乘是:6"
18  "5的阶乘是:120"
19  "10的阶乘是:3628800"
```

优化3：shell 的写法和其他运算表达式。

```
1  #!/bin/bash
2  fun1() {
3    factorial=1
4    #for((i=1;i<=$1;i++))
5    for i in `seq $1`
6    do
7      #factorial=${factorial*$i}
8      #let factorial=factorial*$i
9      let factorial*=i
10   done
11   echo "$1的阶乘是:$factorial"
12 }
13
14 fun1 $1
```

示例3：函数传参 数组传参

1 制作一个简单的阶乘脚本。通过数组给函数传参

```
1  #!/bin/bash
2      #1 先定义一个数组
3  num=(1 2 3)
4      #2 定义一个函数
5  array(){
6    factorial=1
7    for i in $*      #（重点2）
8    do
9      #定义阶乘的公式
10   factorial=${factorial * $i}
11   done
12   echo $factorial
13 }
14      #调用函数使用数组（重点1）
15 array ${num[*]}
```

2 测试成功。

```
1  [root@localhost ~]# ./fun3.sh
2  6
```

3 数组的好处在于，多个数组时传参的效率就增高了。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-8kvUS8ry-1619003860300)(C:\Users\袁威\AppData\Roaming\Typora\typora-user-images\image-20210421190046226.png)]

示例4：函数结果 赋予数组

场景：用户获赠流量包（每人增加5G），结果运算

1 通过函数输出到数组

```
1  #!/bin/bash
2  num=(1 2 3)
3  array(){
4      #local定义变量仅在函数中有效。
5      local j
6      #设置循环的次数，等于索引总数。
7          for i in $*
8          do
9              #定义不同的值乘以5.注意调取值的时候使用的是索引。
10                 outarray[j++]=${i+5}
11             done
12             #输出新的数组。
13             echo "${outarray[*]}"
14         }
15     array ${num[*]}
16
17
```

2 测试脚本成功。

```
1  [root@localhost ~]# bash function_array.sh
2  5 10 15
```

总结：函数不仅可以从数组中调取值，还可以赋予数组值。

三、影响Shell程序的内置命令

1 概览

```
1  :
2  true
3  false
```

```
4 | exit
5 | break
6 | continue
7 | shift
8 | _____
9 |
10 | shift  使位置参数向左移动，默认移动1位，可以使用shift 2
11 | exit  退出整个程序
12 | bre   结束当前循环，或跳出本层循环
13 | continue  忽略本次循环剩余的代码，直接进行下一次循环
```

2 continue和break&shift

需求

1 通过循环脚本，输出如下效果。

```
1 |      A123456789
2 |      B123456789
3 |      ...
4 |      D123456789
```

2 编写循环脚本。

```
1 | #!/bin/bash
2 | for i in {A..D}
3 | do
4 |     echo $i
5 |     for j in {1..9}
6 |     do
7 |         echo $j
8 |     done
9 | done
```

3 测试脚本，观察结果（满意否）

```
1 | [root@localhost ~]# bash for1.sh
2 | A
3 | 1
4 | 2
5 | 3
6 | 4
```

```
7 | 5
8 | 6
9 | 7
10 | 8
11 | 9
12 | B
```

注:因为默认echo输出换行符，使用-n不输出

4 解决换行问题

```
1 | #!/bin/bash
2 | for i in {A..D}
3 | do
4 |     echo -n $i
5 |     for j in {1..9}
6 |     do
7 |         echo -n $j
8 |     done
9 | done
```

5 测试脚本，观察结果（满意否）

```
1 | [root@localhost ~]# bash for1.sh
2 | A123456789B123456789C123456789D123456789
```

6 外循环，添加一条空行语句。

```
1 | #!/bin/bash
2 | for i in {A..D}
3 | do
4 |     echo -n $i
5 |     for j in {1..9}
6 |     do
7 |         echo -n $j
8 |     done
9 | #空行
10 |     echo
11 | done
```

7 再次测试。完成预期

```
1 [root@localhost ~]# bash for1.sh
2 A123456789
3 B123456789
4 C123456789
5 D123456789
```

8 总结：循环嵌套的规则是：外部循环一次，内部循环全部。

9 需求：跳出关于5的循环。

```
1 [root@localhost ~]# bash for1.sh
2 A12346789
3 B12346789
4 C12346789
5 D12346789
```

10 continue登场

```
1 if 判断此次循环是否为5，
2 continue 跳过本次循环。
```

```
1 #!/bin/bash
2 for i in {A..D}
3 do
4     echo -n $i
5     for j in {1..9}
6     do
7         if [ $j -eq 5 ];then
8             continue
9         fi
10        echo -n $j
11    done
12    echo
13 done
```

11 换成break会怎么样呢？

```
1 #!/bin/bash
2 for i in {A..D}
3 do
4     echo -n $i
```

```
5      for j in {1..9}
6      do
7          if [ $j -eq 5 ];then
8              break 2
9          fi
10         echo -n $j
11     done
12     echo
13 done
```

12 测试

```
1 [root@localhost ~]# bash for1.sh
2 A1234
```

13 课后作业，循环嵌套，打印乘法口诀表。

**shift 移动参数**

1 for 循环不定义循环范围，循环取参数作为循环范围。

```
1 #!/bin/bash
2 for i
3 do
4     let sum+=$i
5 done
6 echo "sum : $sum"
```

测试：

```
1 [root@localhost ~]# bash sum.sh
2 sum :
3 [root@localhost ~]# bash sum.sh 1
4 sum : 1
5 [root@localhost ~]# bash sum.sh 2
6 sum : 2
7 [root@localhost ~]# bash sum.sh 3
8 sum : 3
9 [root@localhost ~]# bash sum.sh 1 3
10 sum : 4
```

2 使用while循环，发现停不下来

```
1 #!/bin/bash
2 while [ $# -ne 0 ]
3 do
4     let sum+=${1}
5     echo $sum
6 done
7 echo "sum : $sum"
```

测试:

```
1 [root@localhost ~]# bash sum.sh 1 2
2 根本停不下来。因为循环为真。
```

3 使用shift 移动参数的命令。结果得以实现。

```
1 #!/bin/bash
2 while [ $# -ne 0 ]
3 do
4     let sum+=${1}
5     shift
6 done
7 echo "sum : $sum"
```

测试:

```
1 [root@localhost ~]# bash sum.sh 1 2
2 sum : 3
3 [root@localhost ~]# bash sum.sh 1 2 3
4 sum : 6
```

总结:

```
1 shift 1使参数 左移1位, shift 2 左移2位
```

4 另一个，创建用户的案例。理解参数位移

```
1 #!/bin/bash
2 while [ $# -ne 0 ]
3 do
4     useradd $1
5     echo "$1 is created"
```



```
5 echo $1 $2 created
6 shift
7 done
```

测试:

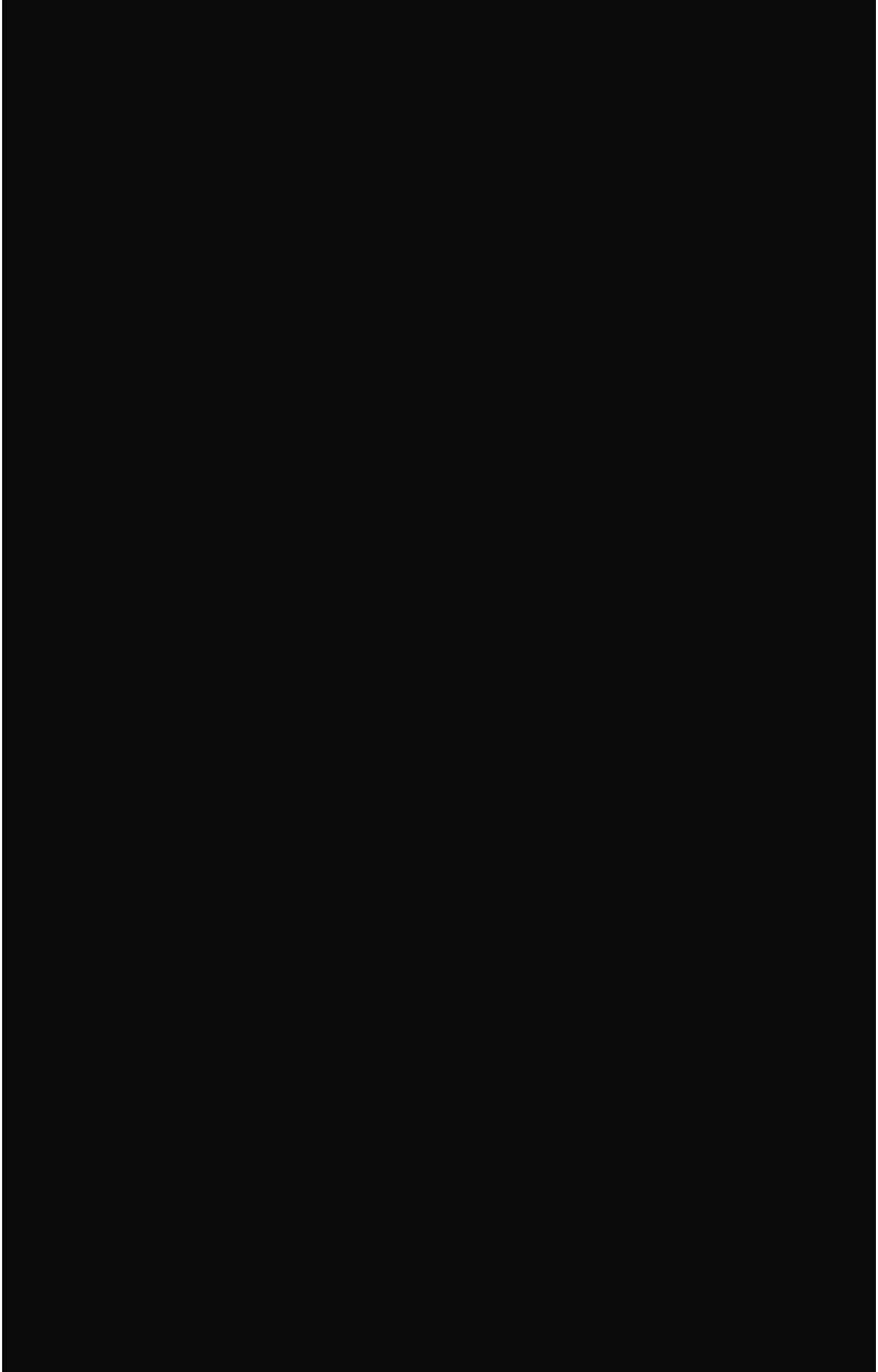
```
1 [root@localhost ~]# bash shift_user.sh aa bb
2 aa is created
3 bb is created
```

🔗 文章知识点与官方知识档案匹配，可进一步学习相关知识

云原生入门技能树 > 基础架构自动编排(Terraform) > 介绍Terraform 13157 人正在系统学习中

LinuxShell自动化运维-课后习题答案.docx	12-17
LinuxShell自动化运维-课后习题答案.docx	
【shell】实现性别统计	weixin_53237028的博客 767
关联数组实现性别统计 [root@server4 ~]# cat sex.txt jack m alice f tom m rose f robin m #!/usr/bin/bash #count sex #by racon 2022-04-22 d...	
shell数组的基本操作_shell 数组操作_宋回嘉乐吗的博客	6-2
数组名[0]="33" 数组名[0]="value" 数组名[1]="value" 数组名[2]="value" 数组的用法 1.获取数组列表 2.获取数组的长度 3.读取数组赋值(根据下...	
shell数组的使用方法_shell 数组使用_爱在梦的另一端的博客	6-7
否则数组中存在包含空格的元素时会按空格将元素拆分成多个 不能将"@"替换为"*",如果替换为"* ",不加双引号时与"@"的表现一致,加双引号时...	
linux 运维自动化脚本,Linux/Unix Shell脚本编程 实现运维工作自动化	weixin_32143245的博客 242
课程内容： Shell脚本编程的技巧永远不会过时：它们可以让Unix充分发挥其真实的潜能。对Unix的用户与系统管理者而言，编写shell脚本是...	
Linux Shell编程及自动化运维实现之数组和函数	yiweii的博客 213
一、数组 简介： 1、数组简介 变量：用一个固定的字符串表示一个不固定的字符串。 数组：用一个固定的字符串，表示多个不固定字符串。 ...	
shell脚本之数组_shell 数组_Tiksty的博客	6-8
shell中,用小括号()来表示数组,数组元素之间用空格来分隔 array1=(1 2 3 4 5) echo \${array1[*]} #输出定义数组的所有元素 echo \${array1[@]}...	
shell数组详解_shell 数组_运维@小兵的博客	6-9
数组是Shell的一种特殊变量,是一组数据的集合,里面的每个数据被称为一个数组元素。 当前Bash仅支持一维索引数组和关联数组,Bash对数组...	
linux自动运维脚本,自动化运维-使用Shell脚本简单实现	weixin_35633340的博客 442
回顾： 1 安装etcd[root@linux-node1 ~]# pip install python-etcd 安装etcd软件2 修改salt-master的配置文件，加配置，并重新启动salt-master[...	
linux运维自动化脚本,linux运维自动化shell脚本小工具	weixin_32123259的博客 1005
linux运维shell 脚本小工具，如要分享此文章，请注明文章出处，以下脚本仅供参考，若放置在服务器上出错，后果请自负1.检测cpu剩余百分...	

<div>SHELL脚本06_shell中100以内含7和7的<b>倍数</b>_CSDN时光的博客</div> <div>NSD <b>SHELL</b> DAY06 目录 1 案例1:使用awk提<b>取</b>文本 1.1 问题 本案例要求使用awk工具完成下列过滤任务: 练习awk工具的基本用法 提<b>取</b>本机...</div>	6-3
<div>shell 中基本运算与<b>数组</b>_MusicDancing的博客</div> <div>shell 中基本运算与<b>数组</b> 1. shell中的基本运算 <b>Shell</b> 支持多种运算符,包括: 1.1 算术运算符 #!/bin/bash a=12 # 加法 b=\$((a+4)) echo"两数之...</div>	5-23
<div>Linux Shell 实现自动化<b>运维</b>实践</div> <div>Linux Shell 实现自动化<b>运维</b>实践，非常详细，建议大家学习下载</div>	07-11
<div>Linux系统配置及<b>运维</b>项目化教程课件—shell编程.pdf</div> <div>Linux系统配置及<b>运维</b>项目化教程课件—shell编程.pdfLinux系统配置及<b>运维</b>项目化教程课件—shell编程.pdfLinux系统配置及<b>运维</b>项目化教程课...</div>	06-24
<div>【shell】 shell <b>数组</b>处理_bdview的博客</div> <div>Shell <b>数组</b>   https://www.runoob.com/linux/linux-shell-array.html 极简 array_name=(value1 value2 ... valuen) array_name[0]=value0 读<b>取数组</b>...</div>	2-14
<div>shell脚本中<b>数组</b>常用操作_公众号-测试生财的博客</div> <div>本文整理了一下<b>shell</b>脚本中关于<b>数组</b>的常用语法。 例子 1)<b>shell</b>中命令执行后返回的结果若为多个,可以将结果保存为<b>数组</b>,比如: #第一种就是...</div>	6-4
<div>自动化<b>运维</b>脚本，<b>Shell</b>版本和Python版本。</div> <div>自动化<b>运维</b>脚本，<b>实现</b>各组件一键化安装和远程<b>运维</b>控制。给<b>运维</b>带来便捷方式。</div>	04-15
<div>SHELL自动化<b>运维</b> .pdf</div> <div>司初创技术团队，没有任何基础设施的情况下，需要搭建一系列code管理以及自动化部署等工具....所以引发了下面一系列的部署过程，历时...</div>	09-06
<div>Shell中<b>函数</b>、字符处理、<b>数组</b>_rick_grace的博客</div> <div>Shell中<b>函数</b>、字符处理、<b>数组</b> <b>函数</b>及中断控制 <b>函数</b>:在<b>Shell</b>脚本中,将一些需重复使用的操作,定义为公共的语句块,即可称为<b>函数</b>。作用:通过...</div>	6-2
<div>Linux Shell自动化<b>运维</b>最佳实践</div> <div>Linux Shell自动化<b>运维</b>最佳实践，这可是心血之作，强烈推荐</div>	04-11
<div>linux经典的自动化<b>运维</b>shell脚本 checkNmp_abiao.rar</div> <div>经典的<b>自动化运维</b>脚本 checkNmp abiao原创，可以<b>实现</b>在nginx宕机时候杀死进程让它自动重启，在php挂掉或者502状态时候，自动检测到...</div>	07-26
<div>Linux Shell <b>数组</b>的创建及使用技巧</div> <div>主要介绍了Linux Shell <b>数组</b>的创建及使用技巧,本文讲解了<b>数组</b>定义、<b>数组读取</b>与赋值以及特殊使用,需要的朋友可以参考下</div>	09-15
<div>Linux<b>运维</b>高手进阶Shell编程最佳实战</div> <div>资源名称：Linux<b>运维</b>高手进阶 <b>Shell</b>编程最佳实战资源截图： 资源太大，传百度网盘了，链接在附件中，有需要的同学自<b>取</b>。</div>	07-23
<div>Linux<b>运维</b>- Shell脚本自动化<b>编程</b>实战-第08章Shell内置命令（2集）.zip</div> <div>Linux<b>运维</b>- Shell脚本自动化<b>编程</b>实战-第08章<b>Shell</b>内置命令（2集）.zip</div>	05-28
<div>day5shell<b>函数</b>和<b>数组</b>shell<b>编程</b>.txt</div> <div>day5shell<b>函数</b>和<b>数组</b>shell<b>编程</b>.txt</div>	03-19
<div>python使用Fabric模块<b>实现自动化运维</b></div>	01-27



简介:Fabric提供了系统基础的操作组件，可以实现本地或远程shell命令，包括：命令执行、文件上传、下载及完整执行日志输出等功能。Fab...

运维常用的 34 个 Linux Shell 脚本.doc

05-24

运维常用的 34 个 Linux Shell 脚本.doc

linux shell自动化运维

最新发布

04-11

Linux shell 自动化运维

是指通过编写 shell 脚本自动化执行一些运维任务，比如自动化部署、自动化备份等。这样可以减轻管理员的工作量， ...

“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

关于我们

招贤纳士

商务合作

寻求报道

 400-660-0108

 kefu@csdn.net

 在线客服

工作时间 8:30-22:00

公安备案号11010502030143

京ICP备19004658号

京网文〔2020〕1039-165号

经营性网站备案信息

北京互联网违法和不良信息举报中心

家长监护

网络110报警服务

中国互联网举报中心

Chrome商店下载

账号管理规范

版权与免责声明

版权申诉

出版物许可证

营业执照

©1999-2023北京创新乐知网络技术有限公司



一荤两素

码龄4年

 暂无认证

30

10万+

128万+

1万+

 等级

原创

周排名

总排名

访问

433

17

44

26

58

积分

粉丝

获赞

评论

收藏









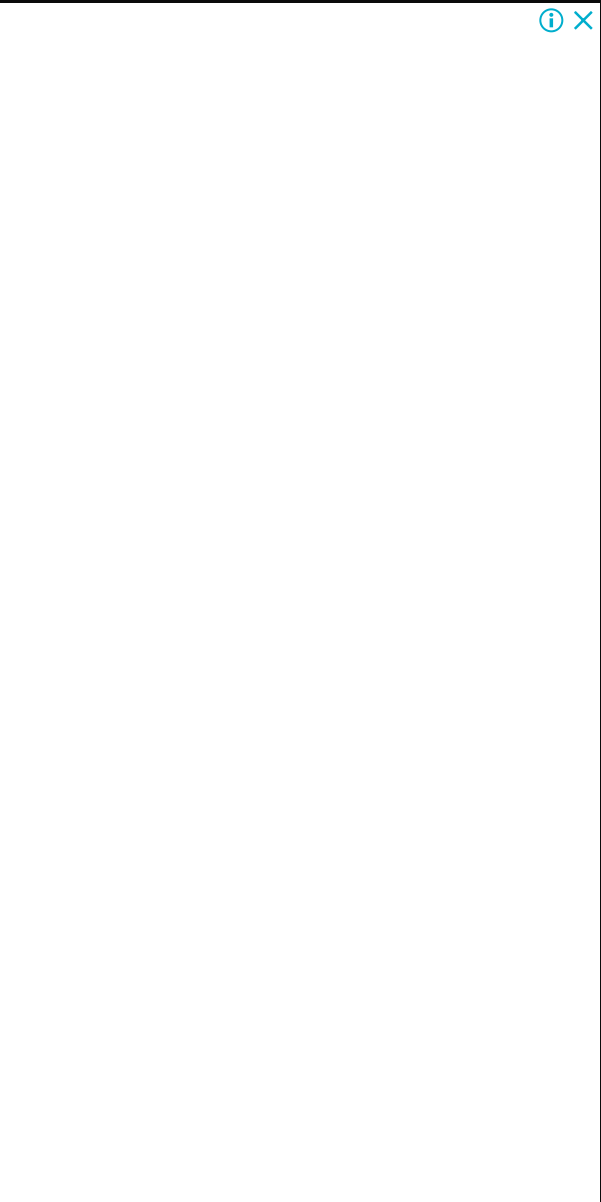


私信

关注

搜博主文章





热门文章

Zabbix自动化监控——监控MySQL

3056

Linux文件管理

2708

Linux中cat /etc/passwd 说明

1477

Zabbix自动化监控——监控Nginx

1414

机械硬盘原理介绍

716

分类专栏



磁盘管理

1篇



mysql集群

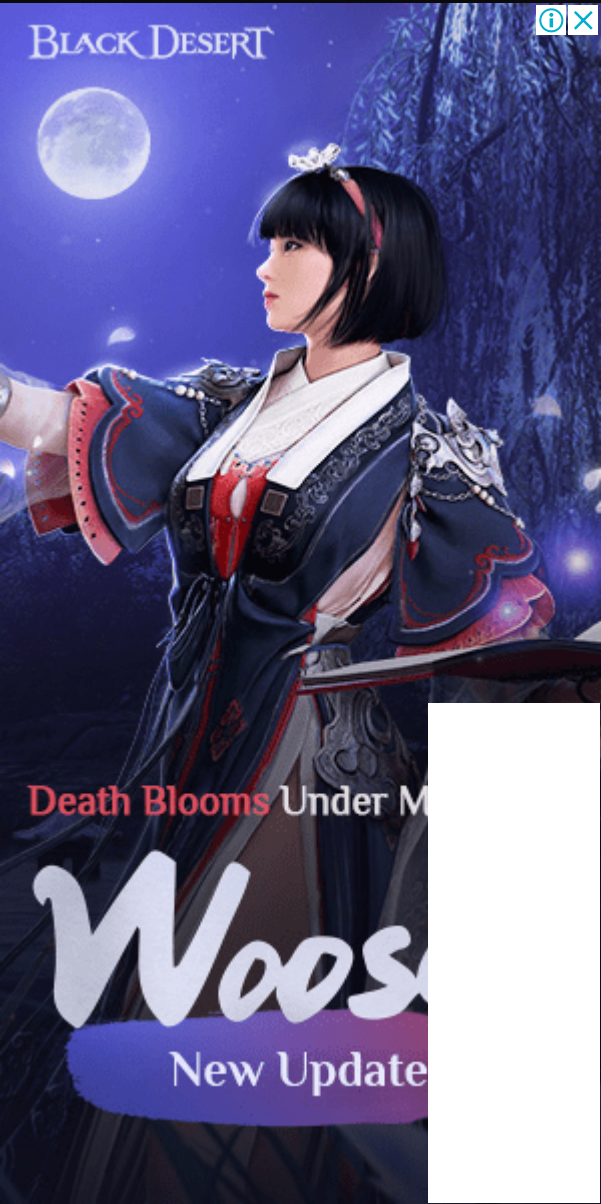
1篇

	主从复制	1篇
	Linux	15篇
	zabbix	3篇
	shell	4篇
▼		

最新评论
Linux Shell编程及自动化运维实现——判断 Cdf（人名）：学习佳作，顺手点赞与关住，期待大佬回访！
Linux Shell编程及自动化运维实现——判断 洛阳泰山: 由浅入深，适合有基础的技术人员。大佬可否给小弟回个赞感谢万分
企业级自动化运维神器/工具Ansible(1) 洛阳泰山: 学习的道路上一同进步，也期待你的关注与支持！
Ansible运维自动化实战——批量部署Nginx 洛阳泰山: 忍不住就是一个赞，写得很棒，欢迎回赞哦~
Ansible运维自动化实战——批量部署Nginx 盼盼编程: 干货满满，很详细，评论占个坑..

您愿意向朋友推荐“博客详情页”吗？
<div></div> <div>强烈不推荐不推荐一般般推荐强烈推荐</div>

最新文章
MySQL集群主从复制
Ansible运维自动化实战——批量部署Nginx
企业级自动化运维神器/工具Ansible(1)
2021年 32篇



目录
Linux Shell编程及自动化运维实现 第4章...
一、数组
二、函数
三、影响Shell程序的内置命令