

# Linux Shell

## 一、Shell小工具

### 1、grep工具（行过滤工具）

*grep是行过滤工具，用于根据关键字进行行过滤*

语法：grep [选项] + '关键字' + 文件名

```
1 # (1) 高亮显示：--color=auto
2 grep --color=auto 'root' password (表示在password文件中搜索含有root的关键字)
```

```
1 # (2) 打印关键字所在的行号：--n
2 grep -n 'root' password
```

```
1 # (3) 打印关键字所在的行号（忽略大小写）：--ni
2 grep -ni 'root' password (在password文件中找root的关键字所在的行号，不管大小写)
```

```
1 # (4) 过滤开头，结尾的行：--^ & --$
2 grep '^root' password (过滤出在password文件中以root开头的关键字的文件)
3 grep 'bash$' password (过滤出在password文件中以bash结尾的文件)
```

```
1 # (5) 和管道符号连用 |
2 env | grep aaa # 在环境变量env中过滤出aaa
```

### 2、cut工具（列截取工具）

*cut是列截取工具，用于列的截取*

语法：cut+选项+文件名

```
1 -c: 以字符为单位进行分割，截取
2 -d: 自定义分隔符号（默认为制表符\t空格）
3 -f: 与-d连用，指定截取哪个区域
```

```
1 # 以: 号作为分割，截取aaa.txt文件中“:”号分割的第1列内容
2 cut -d ':' f1 aaa.txt
```

```
1 # 以: 号作为分割，截取aaa.txt文件中第1, 6, 7列内容
2 cut -d ':' f1,6,7 aaa.txt
```

```
1 # 截取aaa.txt文件中每行第4个字符
2 cut -c4 aaa.txt
```

```
1 #截取aaa.txt文件中每行的第1-4个字符
2 cut -c1-4 aaa.txt
```

```
1 #截取aaa.txt文件中从第5个字符开始的所有字符
2 cut -c5- aaa.txt
```

练习：用小工具查看当前Linux系统的运行级别（运行级别是指系统目前的状态）

```
1 运行级别0：系统停机状态，系统默认运行级别不能设为0，否则不能正常启动
2 运行级别1：单用户工作状态，root权限，用于系统维护，禁止远程登陆
3 运行级别2：多用户状态（没有NFS）
4 运行级别3：完全的多用户状态（有NFS），登陆后进入控制台命令行模式
5 运行级别4：系统未使用，保留
6 运行级别5：X11控制台，登陆后进入图形GUI模式
7 运行级别6：系统正常关闭并重启，默认运行级别不能设为6，否则不能正常启动
```

#### 1、Tips:如何查看系统运行级别

- 使用命令runlevel
- 通过打开文件的方式/etc/inittab

#### 2、如何过滤出运行级别

```
1 #使用runlevel命令时，出现的结果为'N 5'
2 方法1: runlevel | cut -c3 #--结果返回5
3 方法2: runlevel | cut -d ' ' -f2 #--结果返回5
```

## 3、sort工具

sort工具用于排序，将文件的每一行作为一个单位，按每行的首字母升序

```
1 -u:去除重复后排序
2 -r:降序（默认时升序）
3 -n:以数字排序（默认是按字符）
```

## 4、diff工具

diff工具用于逐行比较文件的不同

注意：diff描述两个文件不同的方式是告诉我们如何改变第一个文件之后与第二个文件匹配

```
1 语法: diff [选项] 文件1 文件2
```

常用选项：

选项	含义
-b	不检查空格
-B	不检查空白行

```
1 #使用上下文格式显示两个文件的不同（-c）
2 diff -c aaa.txt bbb.txt
```

```
[root@MissHou ~]# diff -c file1 file2
```

前两行主要列出需要比较的文件名和文件的时间戳；文件名前面的符号\*\*\*表示file1, ---表示file2

```
*** file1      2019-04-16 16:26:05.748650262 +0800
```

```
--- file2      2019-04-16 16:26:30.470646030 +0800
```

```
***** 我是分隔符
```

```
*** 1,6 **** 以***开头表示file1文件, 1,6表示1到6行
```

```
! aaa !表示该行需要修改才与第二个文件匹配
```

```
111
```

```
- hello world -表示需要删除该行才与第二个文件匹配
```

```
222
```

```
- 333 -表示需要删除该行才与第二个文件匹配
```

```
bbb
```

```
--- 1,7 ---- 以---开头表示file2文件, 1,7表示1到7行
```

```
! aaa 表示第一个文件需要修改才与第二个文件匹配
```

```
! hello 表示第一个文件需要修改才与第二个文件匹配
```

```
111
```

```
222
```

```
bbb
```

```
+ 333 表示第一个文件需要加上该行才与第二个文件匹配
```

```
+ world 表示第一个文件需要加上该行才与第二个文件匹配
```

## 二、Shell编程语法类

### 1、shell脚本格式(开头)

```
1 #!/bin/bash
```

### 2、编辑代码

```
1 vim 文件名.sh
2 例如:vim myshell.sh(在myshell.sh里面写代码)
3
4 #在shell文件中编程(这里假设是myshell.sh文件)
5 #!/bin/bash --在shell脚本中#号是注销的符号
6 echo "hello.world" --这里的echo相当于python中的print
```

### 3、shell的执行

```
1 #方法一(实际工作中常用)
2 (1)先给shell文件分配可执行权限
3 chmod 744 myshell.sh
4 (2)使用相对路径调用myshell.sh
5 ./myshell.sh
6
7 #方法二(比较少用,以sh开头,明确是用sh来执行)
8 sh ./myshell.sh
```

练习:

**小试牛刀：**写一个木有灵魂脚本，要求如下：

1. 删除/tmp/目录下的所有文件 `rm -rf /tmp/*`
2. 然后在/tmp目录里创建3个目录，分别是dir1~dir3 `mkdir /tmp/dir{1..3}`
3. 拷贝/etc/hosts文件到刚创建的dir1目录里 `cp /etc/hosts /tmp/dir1`
4. 最后打印"报告首长，任务已于2019-05-05 10:10:10时间完成"内容
5. `echo "报告首长，任务已于$(date +%F %T)"`

```
1  #! /bin/bash
2  rm -rf /tmp/*
3  mkdir /tmp/dir{1..3}
4  cp /etc/hosts /tmp/dir1
5  echo "报告首长，任务已于$(date +%F %T)完成"
6
7  # %F--完整日期格式，等价于 %Y-%m-%d
8  # %T--时间，等于%H:%M:%S
```

## 4、shell的变量

### (1) 变量定义

```
1  # (1) 普通变量定义
2  定义变量: a=100
3  注: 变量定义不能用数字开头
4  引用变量: echo "a=$a"或者echo $a或者echo ${a}
5  删除变量: unset a
6  # (2) 含类型变量定义
7  declare -i A=123 #将变量堪称整数
8  declare -r A=123 #定义自读变量
```

### (2) 变量的分类

1. 本地变量
2. 环境变量
3. 全局变量
4. 系统变量

### (3) 系统变量使用

```
1  #判断上一条指令是否有问题--$?
2  echo $? #返回值为0，无异常；返回值为1，异常
3  #传递参数的个数
4  echo $#
5  #传递的所有参数
6  $*
7  #当前进程的进程号
8  echo $$
```

## 5、运算符

### (1) 普通运算（普通语法只能计算整数之间的四则运算）

```
1  +, -, *, /, % (求余数)
2  #方法一（推荐使用）：
3  $[运算的内容]
4  例如：result=$((2+3)*4)
5
6  #方法二：
7  expr m + n
8
9  #方法三：
10 "$((运算的内容))"
11 例如：result=$((2+3)*4))
```

### (2) 累计运算--Let关键字

```
1  n=10
2  let n=n+1
3  echo $n
```

### (3) 小数四则运算--bc关键字

```
1  echo 1+1.5 |bc
2  或则先bc之后再输入1+1.5、
```

## 6、流程控制

### (1) 条件判断语句

```
1  1、格式：if [ 条件语句 ] fi
2  ###注意：[ ]中的条件语句前后都有空格###
```

#### 字符串比较

```
1  2、比较符
2  #（1）字符串之间的比较，用“=”号
3  例如：if [ "ok"="ok" ]
4          then echo "equal"
5          else echo "not equal"
6          fi
```

#### 数字比较（两种方式）

- 类C风格的数值比较

注意：在(( ))中，=表示赋值；==表示判断

```
[root@server ~]# ((1==2));echo $?  
[root@server ~]# ((1<2));echo $?  
[root@server ~]# ((2>=1));echo $?  
[root@server ~]# ((2!=1));echo $?  
[root@server ~]# ((`id -u`==0));echo $?  
  
[root@server ~]# ((a=123));echo $a  
[root@server ~]# unset a  
[root@server ~]# ((a==123));echo $?
```

```
1 # (2) 数与数之间的比较,用英文缩写  
2 -lt : 小于 means "less than"  
3 -le : 小等于 means "less than equal to"  
4 -eq: 等于 means "equal"  
5 -gt : 大于 means "greater than"  
6 -ge : 大等于 means "greater than equal to"  
7 -ne : 不等于 means "not equal to"  
8  
9 例如:if [ 20 -gt 22 ]  
10     then echo "大于"  
11     else echo "不大于"  
12 fi
```

## 文件比较

```
1 # (3) 判断文件是否存在  
2  
3 例如:  
4     if [ -e /home/abc.txt ]  
5     then echo "存在"  
6     else echo "不存在"  
7     fi
```

## (一) 判断文件类型



判断参数	含义
-e	判断文件是否存在（任何类型文件）
-f	判断文件是否存在并且是一个普通文件
-d	判断文件是否存在并且是一个目录
-L	判断文件是否存在并且是一个软连接文件
-b	判断文件是否存在并且是一个块设备文件
-S	判断文件是否存在并且是一个套接字文件
-c	判断文件是否存在并且是一个字符设备文件
-p	判断文件是否存在并且是一个命名管道文件
-s	判断文件是否存在并且是一个非空文件（有内容）

### 多重条件判断



判断符号	含义	举例
-a 和 &&	逻辑与	[ 1 -eq 1 -a 1 -ne 0 ] [ 1 -eq 1 ] && [ 1 -ne 0 ]
-o 和	逻辑或	[ 1 -eq 1 -o 1 -ne 1 ]

### 小试牛刀:



让用户自己输入字符串，如果用户输入的是hello，请打印world，否则打印“请输入hello”

1. read定义变量
2. if....else...

```
#!/bin/env bash

read -p '请输入一个字符串:' str;
if [ "$str" = 'hello' ];then
    echo 'world'
else echo '请输入hello!'
fi
```

- 1 注意：在 shell 中我们会见到 \$0、\$1、\$2这样的符号，这是什么意思呢？简单来说 \$0 就是你写的 shell 脚本本身的名字（文件名），\$1 是你给你写的shell脚本传的的第一个参数，\$2 是你给你写的 shell 脚本传的第二个参数

## (2)for循环语句

```
1 格式:
2      for 变量 in 值1, 值2, 值3
3      do
4          程序内容
5      done
```

```
1  for i in {1..5}
2  do
3      echo "$i"
4  done
5  #结果显示为1,2,3,4,5
6
7  或则
8  for i in $(seq 5)
9  do
10     echo "$i"
11 done
```

```
1  for i in {1..5..2} #步长为2
2  do
3      echo "$i"
4  done
5  #结果显示为2,4
```

## (3)while循环

```
1 格式:
2      while [ 条件判断式 ]
3      do
4          程序
5      done
```

```
1  while [ 1 -eq 1 ] #此时是死循环
2  do
3      echo "循环"
4  done
```

## 7、读取控制台的输入（类似于python中的input功能）

### 1、基本语法

```
1  read+(选项)+(参数)
2  选项: -p: 读取时的提示内容 -t: 读取时等待的时间 -n: 定义数字长度 m
3  参数: 参数, 指读取值得变量名
4
5  例如:
```



```

6  # (1) 读取控制台输入的一个num值
7  read -p "请输入一个数num=" NUM1
8  echo "你输入的值是 num1=$NUM1"
9  # (2) 输入的数字长度为5
10 read -n 5 -p "请输入一个数字" NUM2
11 echo "你输入的值是 num1=$NUM1"
12 # (3) 读取控制台输入的一个num值, 并且在10秒内输入
13 read -t 10 -p "请输入一个数num" NUM1
14 echo "你输入的值是 num1=$NUM1"
15 # (4) 读取文件(将指定文件中的数据作为变量传出)
16 read -p "请输入ip地址:" IP <ip.txt

```

## 8、自定义函数

```

1  1、定义: [function] funname[()]
2  {
3  action;
4  [return int;]
5  }
6
7  2、调用: funname[值]

```

```

1  例如: 计算输入 (用read方式输入) 两个参数的和
2  定义:
3  function getSum(){
4      SUM=$(( $n1+$n2 ))
5      echo "和是=$SUM"
6  }
7
8  调用:
9  read -p "请输入第一个数n1" n1
10 read -p "请输入第二个数n2" n2
11 getSum $n1 $n2
12
13  注意: 定义和调用的部分都统一写在shell脚本里面

```

## 9、随机数--RANDOM关键字(大写字母)

```

1  #产生0~1之间的随机数
2  echo $[ $RANDOM%2 ]
3  #产生0~2之间的随机数
4  echo $[ $RANDOM%3 ]
5  #打印随机数
6  echo $RANDOM

```

## 常用命令解释

```

1  $[]--进行数学运算
2  $()--进行命令替换 (有限执行权) --和反引号``的用法一样
3  ${}--进行变量替换
4  ``--反引号括起来的字符串被shell解释为命令行, 在执行时, shell首先执行该命令行, 并以它的标准
   输出结果取代整个反引号 (包括两个反引号) 部分, 也就是把反引号中命令行执行的结果赋值给变量

```

## 10、crontab-e命令

例如\*/1 \* \* \* \* mytask1.sh 表示每隔1分钟执行一次mytask1.sh脚本文件，现做以下说明

*	*	*	*	*
minute	hour	day	month	week
表示分钟，可以是0到59之间的任何整数	表示小时，可以是0到23之间的任何整数	表示日期，可以是1到31之间的任何整数	表示月份，可以是1到12之间的任何整数	表示星期几，可以是0到6之间的任何整数，这里的0代表星期日

- 1
- 1、星号\*,代表所有可能的值，例如month字段如果是星号，则表示在满足其它字段的制约条件后每月都执行该命令操作；
- 2
- 2、逗号(,)：可以用逗号隔开的值指定一个列表范围，例如，“1,2,5,7,8,9”；
- 3
- 3、中杠(-)：可以用整数之间的中杠表示一个整数范围，例如“2-6”表示“2,3,4,5,6”；
- 4
- 4、正斜线(/)：可以用正斜线指定时间的间隔频率，例如“0-23/2”表示每两小时执行一次。同时正斜线可以和星号一起使用，例如\*/10，如果用在minute字段，表示每十分钟执行一次。

## 三、文件编辑器

### 1、sed

一行一行读取文件内容并按照要求进行处理，把处理前、后的结果都输出到屏幕.好处是在不打开编辑器的情况下就编辑文件的内容

#### (1) 语法

- 1
- sed 选项 '处理动作' 文件名

## • 常用选项

选项	说明	备注
-e	进行多项(多次)编辑	
-n	取消默认输出	不自动打印模式空间
-r	使用扩展正则表达式	
-i	原地编辑 (修改源文件)	
-f	指定sed脚本的文件名	

动作	说明	备注
'p'	打印	
'i'	在指定行之前插入内容	类似vim里的大写O
'a'	在指定行之后插入内容	类似vim里的小写o
'c'	替换指定行所有内容	
'd'	删除指定行	

## (2) 举例

```

1 | 打印
2 | sed -n 'p' a.txt    #打印a.txt文件所有数据在屏幕上，同时取消前后结果同时打印（只打印结果数据）
3 | sed -n '2p' a.txt   #打印a.txt文件第2行

```

```

1 | 插入
2 | sed '2iHello world' a.txt    #在a.txt文件中的第2行前面插入Hello world
3 | sed '2aHello world' a.txt    #在a.txt文件中的第2行之后插入Hello world

```

```

1 | 替换--s表示搜索（search）
2 | sed -n 's/root/ROOT/p' a.txt #在a.txt文件中搜索root，将root替换成ROOT（替换每行符合条件的第一个），并打印出来
3 |
4 | sed -n 's/root/ROOT/gp' a.txt #在a.txt文件中搜索root，将root替换成ROOT(全局替换)，并打印出来
5 |
6 | sed -n 's/root//gp' a.txt    #在a.txt文件中搜索root，将root替换成空（全局替换），并打印出来
7 |
8 | sed -ne '/^home/p' -ne '/^hmo=/=' #表示多次操作（e）先将每行开头是home得行打印出来，然后再多次操作。将每行是home得行号打印出来

```

语法: sed 选项 's/搜索的内容/替换的内容/动作' 需要处理的文件

其中, s表示search搜索; 斜杠/表示分隔符, 可以自己定义; 动作一般是打印p和全局替换g

```
[root@server ~]# sed -n 's/root/ROOT/p' 1.txt
[root@server ~]# sed -n 's/root/ROOT/gp' 1.txt
[root@server ~]# sed -n 's/^#//gp' 1.txt
[root@server ~]# sed -n 's@/sbin/nologin@itcast@gp' a.txt
[root@server ~]# sed -n 's/\/sbin\/nologin/itcast/gp' a.txt
[root@server ~]# sed -n '10s#/sbin/nologin#itcast#p' a.txt
uucp:x:10:14:uucp:/var/spool/uucp:itcast
[root@server ~]# sed -n 's@/sbin/nologin@itcastheima@p' 2.txt
注意: 搜索替换中的分隔符可以自己指定
```

命令	解释	备注
r	从另外文件中读取内容	
w	内容另存为	
&	保存查找串以便在替换串中引用	和\(\)相同
=	打印行号 <b>重要</b>	
!	对所选行以外的所有行应用命令, 放到行数之后	<b>I</b>
q	退出	

## 四、awk使用

awk的作用: 处理文件和数据, 是一种编程语言

- 1、统计数据, 例如网站的访问量, 访问IP量等等
- 2、支持条件判断、for\while循环
- 3、可以用空格作为分隔符, 这个是cut做不到的

### 1、语法结构

- 1 awk '条件{动作}' 文件名
- 2 注意: awk的语法结构和sed的一样, 当引用shell变量需用双引号引起来

### 2、案例

- 1 #提取第四列
- 2 [xiongyuan@hadoop100 ~]\$ awk '{print \$4}'

```
name    age    sex    class
xiong   1      m      1
yuan    2      f      2
xyuan   2      3      4
```

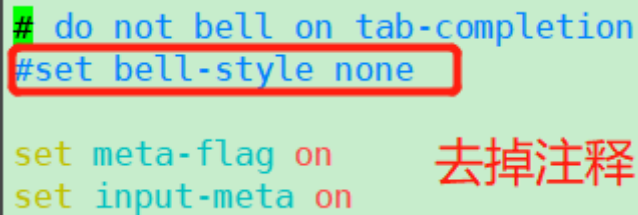
```
1 #当第4列的值为2时，打印第一列的值
2 [xiongyuan@hadoop100 ~]$ awk '$4 ==2 {print $1}' xiongyuan2.txt
```

```
1 #当行号为3的时候，打印第一列
2 [xiongyuan@hadoop100 ~]$ awk 'NR==3 {print $1}' xiongyuan2.txt
```

## 五、shell小技巧

### (一)关闭提示音

```
1 进入 vim /etc/inputrc 然后重启
```



```
# do not bell on tab-completion
#set bell-style none

set meta-flag on
set input-meta on
```

去掉注释

### (二) 设置判断shell脚本中输出是空值还是为定义变量

```
1 set -u
```

### (三) 位置参数变量

```
1 #位置参数的缺点，别人看脚本很困难，只有脚本编写者本人对脚本比较清楚
2 $0 脚本本身
3 $1-9 触发脚本时输入的第1-9个参数
4 $* 脚本所有的参数
5 $@ 脚本所有的参数，把所有参数逐一赋值给$@ (和for循环配合使用)
6 $# 脚本参数的数量
7 $? 返回值 (在执行$?之前最后一次执行的命令的返回状态，是自定义的)
8 注意：关于$?, 返回值为127，默认是linux的上一条命令执行是出现错误
```

## 六、正则表达式

正则表达式用来在文件中匹配字符串

通配符用来匹配文件名

```
1 *: 在某个字符后面加*, 同时写出多个这个字符, 表示这个字符至少出现的次数
2 grep "aaa*" text.txt #表示对a进行规定, 规定a连续出现的次数不少于2次
3 ^: 匹配开头字符
4 grep "^x" text.txt
5 $: 匹配结尾字符
6 grep "d$" text.txt
7 [: 匹配[]中任意一个字符
8 grep "x[abi]ong" text.txt #表示匹配要么xiong, xaong, xbong中的一个
9 [A-Z]: 匹配[]中的任意大写字符
10 grep "x[A-Z]ong" text.txt
```

```
11 [a-z]: 匹配[]中任意小写字符
12 grep "x[a-z]ong" text.txt
13 [0-9]: 匹配[]中任意数字
14 grep "x[0-9]ong" text.txt
15 [{\n\\}]: n里面写对应的数字, 表示后面匹配n个数字
```