shell脚本

shell介绍

什么是shell

> 用户和kernel沟通的桥梁，既是一种命令式语言，又是一种解释性语言

shell的功能

1. 命令行解释功能
2. 启动程序
3. 输入输出重定向
4. 管道连接
5. 文件名置换
6. 变量维护
7. 环境控制
8. shell编程

shell语法

shell脚本就是将完成一个任务的所有命令按照执行的先后顺序，自上而下写到一个文本文件中，然后给与执行权限。

```
#!/bin/bash

#Author: wangml
#Created Time: 2121/09/22 16:00
#Script Description: install nginx

yum -y install wget gcc pcre-devel zlib-devel
wget http://nginx.org/download/nginx-1.16.0.tar.gz
tar xf nginx-1.16.0.tar.gz
cd nginx-1.16.0
./configure --prefix=/usr/local
make
make install
```

shell脚本格式

> shell脚本开头必须指定脚本运行环境 #!/bin/bash
>
> 注释使用 # 开头

shell脚本的运行

> 1. chmod 700 name.sh 赋予文件执行权限再运行 ./name.sh
> 2. bash name.sh

shell中的特殊符号

```
~:                家目录     # cd ~ 代表进入用户家目录
!:                执行历史命令    !! 执行上一条命令
$:                变量中取内容符
+ - * 、 %:        对应数学运算   加 减 乘 除 取余数
&:                后台执行
*:                星号是shell中的通配符   匹配所有有
?:                问号是shell中的通配符   匹配除回车以外的一个字符
;:                分号可以在shell中一行执行多个命令，命令之间用分号分割
|:                管道符 上一个命令的输出作为下一个命令的输入    cat filename | grep "abc"
\:                转义字符
``:               反引号 命令中执行命令     echo "today is `date +%F`"
' ':              单引号，脚本中字符串要用单引号引起来，但是不同于双引号的是，单引号不解释变量
" ":              双引号，脚本中出现的字符串可以用双引号引起来
```

shell中管道的使用

> 管道 | 将一个命令的输出当做输入交给另一个命令

```
(base) [root@CentOS-wangml shell]# ll
total 44
-rw-r--r-- 1 root root  19 Sep 24 09:39 abc.txt
-rwx------ 1 root root 175 Sep 23 14:35 array.sh
-rwx------ 1 root root 267 Sep 23 14:47 ass_array.sh
-rwx------ 1 root root 170 Sep 22 20:00 countdown.sh
-rwx------ 1 root root 151 Sep 22 17:20 disk_partition.sh
-rwx------ 1 root root 257 Sep 24 10:01 for.sh
-rwx------ 1 root root 498 Sep 24 09:39 if_else.sh
-rwx------ 1 root root 319 Sep 24 10:39 monitor.sh
-rwx------ 1 root root 288 Sep 22 16:17 nginx_install.sh
-rwx------ 1 root root 234 Sep 22 20:29 read_command.sh
-rwx------ 1 root root  67 Sep 22 17:06 temp.sh
(base) [root@CentOS-wangml shell]# ll | wc -l
12
```

shell重定向

命令	说明
command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。
n > file	将文件描述符为 n 的文件重定向到 file。
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file。
n >& m	将输出文件 m 和 n 合并。
n <& m	将输入文件 m 和 n 合并。
<< tag	将开始标记 tag 和结束标记 tag 之间的内容作为输入。

shell格式化输出

echo

> -n 输出后不自动换行
>
> -e 出现下列字符特别处理
>
> 1. \a 警告声
> 2. \b 删除最后一个字符
> 3. \t tab

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: echo 实现倒计时

echo

for time in `seq 9 -1 0`;do
    echo -n -e "\b$time"
    sleep 1
done

echo
```

shell基本输入

read

> 选项：
>
> 1. -p 打印信息
> 2. -t 限定时间
> 3. -s 不回显
> 4. -n 输入字符个数限制

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

```
#Author: wangml
#Created Time:
#Script Description: read命令

clear

#echo -n -e "Login: "
#read account
read -p "Login: " account
echo -n -e "Passwd: "
read -s -t5 -n6 pd

echo
echo "account: $account, password: $pd"
```

变量

注意

> 注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：
>
> - 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
> - 中间不能有空格，可以使用下划线 _。
> - 不能使用标点符号。
> - 不能使用bash里的关键字（可用help命令查看保留关键字）。

使用变量

> 通过 $VALUE 获取变量内容

变量类型

> 运行shell时，会同时存在三种变量：
>
> - **1) 局部变量** 局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量。
> - **2) 环境变量** 所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量。
> - **3) shell变量** shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行

只读变量

> readonly VALUE声明只读变量

删除变量

> unset VALUE

数组

基本数组

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 数组的介绍

ARRAY1=('a' 'b' 'c' 'd')

echo ${ARRAY1[2]}

ARRAY1[4]='E'
ARRAY1[5]='F'
echo ${ARRAY1[4]}
```

关联数组

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 关联数组的使用

declare -A ASS_ARRAY1
declare -A ASS_ARRAY2

ASS_ARRAY1[name]='libai'
ASS_ARRAY1[age]=54

ASS_ARRAY2=([name]='dufu' [age]=64)

echo ${ASS_ARRAY1[name]}

echo ${ASS_ARRAY2[name]}
```

流程控制

数据比较

- File type tests:: -[bcdfhLpSt]
- Access permission tests:: -[gkruwxOG]
- File characteristic tests:: -e -s -nt -ot -ef
- String tests:: -z -n = == !=
- Numeric tests:: -eq -ne -lt -le -gt -ge
- Connectives for test:: ! -a -o

数学的比较运算

```
INTEGER1 -eq INTEGER2
==

INTEGER1 -ge INTEGER2
>=
```

```
INTEGER1 -gt INTEGER2
>

INTEGER1 -le INTEGER2
<=

INTEGER1 -lt INTEGER2
<

INTEGER1 -ne INTEGER2
!=
```

字符串的比较

1. ==
2. !=
3. -n 检查字符串长度是否大于0
4. -z 检测字符串长度是否为0

文件的比较

```
   FILE1 -nt FILE2
          FILE1 is newer (modification date) than FILE2

   FILE1 -ot FILE2
          FILE1 is older than FILE2

   -b FILE
          FILE exists and is block special

   -c FILE
          FILE exists and is character special

   -d FILE
          FILE exists and is a directory

   -e FILE
          FILE exists

   -f FILE
          FILE exists and is a regular file

   -g FILE
          FILE exists and is set-group-ID

   -G FILE
          FILE exists and is owned by the effective group ID

   -h FILE
          FILE exists and is a symbolic link (same as -L)

   -k FILE
          FILE exists and has its sticky bit set

   -L FILE
          FILE exists and is a symbolic link (same as -h)
```

```
   -O FILE
          FILE exists and is owned by the effective user ID

   -p FILE
          FILE exists and is a named pipe

   -r FILE
          FILE exists and read permission is granted

   -s FILE
          FILE exists and has a size greater than zero

   -S FILE
          FILE exists and is a socket

   -t FD  file descriptor FD is opened on a terminal

   -u FILE
          FILE exists and its set-user-ID bit is set
```

逻辑运算

> 1. &&
> 2. ||
> 3. !

if else

格式直接看代码

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 流程控制

CNT=$1

if [ $CNT -gt 5 ]
then
    echo "CNT > 10"
fi

# 如果在当前目录下没有abc.txt文件 则创建
if [ ! -e ./abc.txt ]
then
    touch ./abc.txt
    echo "create abc.txt"
else
    echo "abc.txt is existed" > ./abc.txt
fi


# (())使用数学运算
if (( 100%3+1>10 ));then
    echo "yes"
```

```
else
    echo "no"
fi

# [[]] 使用通配符
if [[ "abcnews" == abc* ]];then
    echo "yes"
else
    echo "no"
fi
```

for

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: for循环

for var in 1 2 3 4 5 6 7 8 9
do
    echo $var
    #sleep 1
done


for var in `seq 1 9`
do
    echo $var
    #sleep 1
done


for (( i=1;i<10;i++ ))
do
    echo $i
    #sleep 1
done
```

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 定期查看主机是否可以ping通某个ip

for ((;;))
do
    ping -c1 $1 &>/dev/null
    if [ $? -eq 0 ]
    then
        echo "`date +"%F %H:%M:%S"`: $1 is UP"
    else
        echo "`date +"%F %H:%M:%S"`: $1 is DOWN"
    fi

    sleep 5
```

```
done
```

while

```
(base) [root@CentOS-wangml shell]# bash while.sh 3
3
2
1
(base) [root@CentOS-wangml shell]# cat while.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: while的使用

NUM=$1

while [ $NUM -gt 0 ]
do
    echo $NUM
    NUM=$(($NUM-1))
done
```

break和continue

```
(base) [root@CentOS-wangml shell]# bash loop_control.sh 10
CNT = 1
CNT = 2
CNT = 4
CNT = 5
(base) [root@CentOS-wangml shell]# cat loop_control.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 循环控制

NUM=$1
CNT=1

while [ $CNT -le $NUM ]
do
    if [ $CNT -eq 3 ]
    then
        CNT=$(($CNT+1))
        continue
    fi

    echo "CNT = $CNT"
    CNT=$(($CNT+1))

    if [ $CNT -gt 5 ]
    then
        break
    fi
done
```

until

与while正好相反，until是条件为假开始执行，条件为真停止执行

```
(base) [root@CentOS-wangml shell]# bash until.sh -5
-5
-4
-3
-2
-1
0
(base) [root@CentOS-wangml shell]# cat until.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: until的使用

NUM=$1

until [ $NUM -gt 0 ]
do
    echo $NUM
    NUM=$(($NUM+1))
done
```

case

```
(base) [root@CentOS-wangml shell]# bash case.sh root
You are administrator...
(base) [root@CentOS-wangml shell]# bash case.sh user
You are user..
(base) [root@CentOS-wangml shell]# bash case.sh other
Who you are...
(base) [root@CentOS-wangml shell]# cat case.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: case的使用

case $1 in
    "root")
        echo "You are administrator..."
    ;;
    "user")
        echo "You are user.."
    ;;
    *)
        echo "Who you are..."
    ;;
esac
```

***)** 在所有条件都不满足时执行

shell函数

函数定义格式

```
(base) [root@CentOS-wangml shell]# bash function.sh
Apache start ...            [OK]
Apache stop ...             [FAIL]
Apache start ...            [OK]
Apache start ...            [OK]
Apache start ...            [OK]
Apache stop ...             [FAIL]
(base) [root@CentOS-wangml shell]# cat function.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 函数的使用

# 定义函数
start () {
    echo "Apache start ...            [OK]"
    #return 0
}

function stop {
    echo "Apache stop ...             [FAIL]"
}

# 调用函数
start
# ...
stop

start
start
start

stop
```

nginx服务管理示例

> nginx服务启动失败，80端口被占用问题如何解决?
>
> https://www.cnblogs.com/yiyi17/p/10276045.html
>
> https://blog.csdn.net/yufeng_lai/article/details/88819981

```
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: Nginx服务管理脚本

# Nginx安装目录与执行文件所在地址 参考之前nginx的安装脚本
nginx_install_doc=/usr/local
```

```
nginxd=$nginx_install_doc/sbin/nginx
pid_file=$nginx_install_doc/logs/nginx.pid

# 参考/etc/init.d/network
# Source function library.
if [ -f /etc/init.d/functions ];then
    . /etc/init.d/functions
else
    echo "not found file /etc/init.d/functions"
    exit
fi

start () {
    # 如果nginx没有启动 则启动
    if [ -f $pid_file ];then
        nginx_process_id=`cat $pid_file`
        nginx_process_num=`ps aux | grep $nginx_process_id | grep -v "grep" | wc
-l`
    fi

    if [ -f $pid_file ] && [ $nginx_process_num -ge 1 ];then
        echo "nginx running ..."
    else
        if [ -f $pid_file ] && [ $nginx_process_num -lt 1 ];then
            rm -f $pid_file
            echo "nginx start `daemon $nginxd`"
            # 启动nginx 也可以用下面这种方式
            # action "nginx start" $nginxd
        fi
        echo "nginx start `daemon $nginxd`"
    fi
}

stop () {
    if [ -f $pid_file ];then
        nginx_process_id=`cat $pid_file`
        nginx_process_num=`ps aux | grep $nginx_process_id | grep -v "grep" | wc
-l`
    fi
    # 如果nginx已经启动 则停止服务
    if [ -f $pid_file ] && [ $nginx_process_num -ge 1 ];then
        action "nginx stop" killall -s QUIT nginx
        rm -f $pid_file
    else
        action "nginx stop" killall -s QUIT nginx 2 > /dev/null
    fi
}

restart () {
    stop
    sleep 1
    start
}

reload () {
    if [ -f $pid_file ];then
        nginx_process_id=`cat $pid_file`
```

```
        nginx_process_num=`ps aux | grep $nginx_process_id | grep -v "grep" | wc
-l`
    fi

    if [ -f $pid_file ] && [ $nginx_process_num -ge 1 ];then
        action "nginx reload" killall -s HUP nginx
    else
        action "nginx reload" killall -s HUP nginx 2 > /dev/null
    fi
}

status () {
    if [ -f $pid_file ];then
        nginx_process_id=`cat $pid_file`
        nginx_process_num=`ps aux | grep $nginx_process_id | grep -v "grep" | wc
-l`
    fi

    if [ -f $pid_file ] && [ $nginx_process_num -ge 1 ];then
        echo "nginx running"
    else
        echo "nginx stop"
    fi
}

case $1 in
    start) start;;
    stop) stop;;
    restart) restart;;
    reload) reload;;
    status) status;;
    *) echo "USAGE: $0 start|stop|restart|reload|status";;
esac
```

正则表达式

特殊字符

位置标记

正则表达式	描述	示例
`^`	指定了匹配正则表达式的文本必须起始于字符串的首部	`^tux` 能够匹配以 `tux` 起始的行
`$`	指定了匹配正则表达式的文本必须结束于目标字符串的尾部	`tux$` 能够匹配以 `tux` 结尾的行

当 ^ $ 一起使用时表示精确匹配

> ^ab$ 表示只匹配字符串 ab

标识符

正则表达式	描述	示例
`A` 字符	正则表达式必须匹配该字符	A能够匹配字符A
`.`	匹配任意一个字符	`Hack.` 能够匹配 `Hackl` 和 `Hacki`，但是不能匹配 `Hackl2` 或 `Hackil`，它只能匹配单个字符
`[]`	匹配中括号内的任意一个字符。中括号内可以是一个字符组或字符范围	`coo[kl]` 能够匹配 `cook` 或 `cool`，`[0-9]` 匹配任意单个数字
`[^]`	匹配不在中括号内的任意一个字符。中括号内可以是一个字符组或字符范围	`9[^01]` 能够匹配 `92` 和 `93`，但是不匹配 `91` 和 `90`；`A[^0-9]` 匹配 `A` 以及随后除数字外的任意单个字符

数量修饰符

正则表达式	描述	示例
`?`	匹配之前的项1次或0次	`colou?r` 能够匹配 `color` 或 `colour`，但是不能匹配 `colouur`
`+`	匹配之前的项1次或多次	`Rollno-9+` 能够匹配 `Rollno-99` 和 `Rollno-9`，但是不能匹配 `Rollno-`
`*`	匹配之前的项0次或多次	`co*l` 能够匹配 `cl`、`col` 和 `coool`
`{n}`	匹配之前的项*n*次	`[0-9]{3}` 能够匹配任意的三位数，`[0-9]{3}` 可以扩展为 `[0-9][0-9][0-9]`
`{n,}`	之前的项至少需要匹配*n*次	`[0-9]{2,}` 能够匹配任意一个两位或更多位的数字
`{n,m}`	之前的项所必须匹配的最小次数和最大次数	`[0-9]{2,5}` 能够匹配两位数到五位数之间的任意一个数字

其他

正则表达式	描述	示例
`()`	将括号中的内容视为一个整体	`ma(tri)?x` 能够匹配 `max` 或 `matrix`
`|`	指定了一种选择结构，可以匹配 `|` 两边的任意一项	`Oct (1st | 2nd)` 能够匹配 `Oct 1st` 或 `Oct 2nd`
`\`	转义字符可以转义之前介绍的特殊字符	`a\.b` 能够匹配 `a.b`，但不能匹配 `ajb`。因为 `\` 忽略了 `.` 的特殊意义

POSIX字符

POSIX字符	含义
[:alnum:]	匹配 任意数字字母字符 0-9 a-z A-Z
[:alpha:]	匹配 任意字母， 大写或小写 a-z A-Z
[:digit:]	匹配 任意数字 0-9
[:graph:]	匹配 非空字符， 非空格控制字符
[:lower:]	匹配 小写字母 a-z
[:upper:]	匹配 大写字母 A-Z
[:ctrl:]	匹配 控制字符
[:print:]	匹配非空字符， 包括空格
[:punct:]	匹配标点符号
[:blank:]	匹配空格和TAB字符
[:xdigit:]	匹配16进制数字
[:space:]	匹配所有空白字符，新行、空格、制表符

shell对文件的操作

sed

> 行编辑器，相对于vim类编辑器，在操作大文件时，避免了一下子将打开大文件占用大量内存

sed命令

sed 可依照脚本的指令来处理、编辑文本文件。

Sed 主要用来自动编辑一个或多个文件、简化对文件的反复操作、编写转换程序等

语法

```
sed [option]'{command}[flags]'[文本文件]
```

参数

- -e或--expression= 以选项中指定的script来处理输入的文本文件。
- -f<script文件>或--file=<script文件> 以选项中指定的script文件来处理输入的文本文件。
- -h或--help 显示帮助。
- -n或--quiet或--silent 仅显示script处理后的结果。
- -V或--version 显示版本信息。
- -n抑制自动输出
- -i编辑文件内容
- -r使用扩展正则表达式
- ！ 取反（跟在模式条件后与shell有所区别）

命令

- a： 新增， a 的后面可以接字串，而这些字串会在新的一行出现(目前的下一行)～
- c： 取代， c 的后面可以接字串，这些字串可以取代 n1,n2 之间的行！
- d： 删除，因为是删除啊，所以 d 后面通常不接任何咚咚；
- i： 插入， i 的后面可以接字串，而这些字串会在新的一行出现(目前的上一行)；
- p： 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～
- s： 取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正规表示法！例如 1,20s/old/new/g 就是啦！
- y：转换

```
(base) [root@CentOS-wangml shell]# cat test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed 'aHello World' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
Hello World
2 the quick brown fox jumps over the lazy dog.
Hello World
3 the quick brown fox jumps over the lazy dog.
Hello World
4 the quick brown fox jumps over the lazy dog.
Hello World
5 the quick brown fox jumps over the lazy dog.
Hello World
(base) [root@CentOS-wangml shell]# sed '1aHello World' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
Hello World
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed '2,4aHello World' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
Hello World
3 the quick brown fox jumps over the lazy dog.
Hello World
4 the quick brown fox jumps over the lazy dog.
Hello World
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed '3cHello World' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
Hello World
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed '3,4cHello World' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
Hello World
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed '2,4d' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
```

```
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed '1,3iHello World' test_sed.txt
Hello World
1 the quick brown fox jumps over the lazy dog.
Hello World
2 the quick brown fox jumps over the lazy dog.
Hello World
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# sed -r '2,4s/dog/cat/' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy cat.
3 the quick brown fox jumps over the lazy cat.
4 the quick brown fox jumps over the lazy cat.
5 the quick brown fox jumps over the lazy dog.
```

awk

> awk是一种可以处理数据、产生格式化报表的语言，功能十分强大。awk认为文件中的每一行是一条记录，记录与记录的分隔符为换行符，每一列是一个字段，字段与字段的分隔符默认是一个或多个空格或tab制表符。

```
awk [options][BEGIN]{program}[END][file]
```

选项参数说明：

- **-F** fs or --field-separator fs
  指定输入文件折分隔符，fs是一个字符串或者是一个正则表达式，如-F:。
- -v var=value or --asign var=value
  赋值一个用户定义变量。
- -f scripfile or --file scriptfile
  从脚本文件中读取awk命令。
- -mf nnn and -mr nnn
  对nnn值设置内在限制，-mf选项限制分配给nnn的最大块数目；-mr选项限制记录的最大数目。这两个功能是Bell实验室版awk的扩展功能，在标准awk中不适用。
- -W compact or --compat, -W traditional or --traditional
  在兼容模式下运行awk。所以gawk的行为和标准的awk完全一样，所有的awk扩展都被忽略。
- -W copyleft or --copyleft, -W copyright or --copyright
  打印简短的版权信息。
- -W help or --help, -W usage or --usage
  打印全部awk选项和每个选项的简短说明。
- -W lint or --lint
  打印不能向传统unix平台移植的结构的警告。
- -W lint-old or --lint-old
  打印关于不能向传统unix平台移植的结构的警告。
- -W posix
  打开兼容模式。但有以下限制，不识别：/x、函数关键字、func、换码序列以及当fs是一个空格时，将新行作为一个域分隔符；操作符**和**=不能代替^和^=；fflush无效。
- -W re-interval or --re-inerval
  允许间隔正则表达式的使用，参考(grep中的Posix字符类)，如括号表达式[[:alpha:]]。
- -W source program-text or --source program-text
  使用program-text作为源代码，可与-f命令混用。

- -W version or --version
  打印bug报告信息的版本。

运算符

运算符	描述
= += -= *= /= %= ^= **=	赋值
?:	C条件表达式
||	逻辑或
&&	逻辑与
~ 和 !~	匹配正则表达式和不匹配正则表达式
< <= > >= != ==	关系运算符
空格	连接
+ -	加，减
* / %	乘，除与求余
+ - !	一元加，减和逻辑非
^ ***	求幂
++ --	增加或减少，作为前缀或后缀
$	字段引用
in	数组成员

内建变量

变量	描述
$n	当前记录的第n个字段，字段间由FS分隔
$0	完整的输入记录
ARGC	命令行参数的数目
ARGIND	命令行中当前文件的位置(从0开始算)
ARGV	包含命令行参数的数组
CONVFMT	数字转换格式(默认值为%.6g)ENVIRON环境变量关联数组
ERRNO	最后一个系统错误的描述
FIELDWIDTHS	字段宽度列表(用空格键分隔)
FILENAME	当前文件名
FNR	各文件分别计数的行号
FS	字段分隔符(默认是任何空格)
IGNORECASE	如果为真，则进行忽略大小写的匹配
NF	一条记录的字段的数目
NR	已经读出的记录数，就是行号，从1开始
OFMT	数字的输出格式(默认值是%.6g)
OFS	输出字段分隔符，默认值与输入字段分隔符一致。
ORS	输出记录分隔符(默认值是一个换行符)
RLENGTH	由match函数所匹配的字符串的长度
RS	记录分隔符(默认是一个换行符)
RSTART	由match函数所匹配的字符串的第一个位置
SUBSEP	数组下标分隔符(默认值是/034)

```
(base) [root@CentOS-wangml ~]# cd /home/lighthouse/study/linux/shell
(base) [root@CentOS-wangml shell]# cat test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# awk '{print $0}' test_sed.txt
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# awk '{print $NF}' test_sed.txt
dog.
dog.
```

```
dog.
dog.
dog.
(base) [root@CentOS-wangml shell]# awk '{print $3}' test_sed.txt
quick
quick
quick
quick
quick
(base) [root@CentOS-wangml shell]# awk 'NR==3{print $0}' test_sed.txt
3 the quick brown fox jumps over the lazy dog.
(base) [root@CentOS-wangml shell]# cp /etc/pa
pam.d/   passwd   passwd-
(base) [root@CentOS-wangml shell]# cp /etc/passwd ./
(base) [root@CentOS-wangml shell]# ls
abc.txt   ass_array.sh  countdown.sh       for.sh       if_else.sh
monitor.sh        nginx_start.sh  read_command.sh  test_sed.txt  while.sh
array.sh  case.sh       disk_partition.sh  function.sh  loop_control.sh
nginx_install.sh  passwd          temp.sh          until.sh
(base) [root@CentOS-wangml shell]# cat passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
libstoragemgmt:x:998:997:daemon account for
libstoragemgmt:/var/run/lsm:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:997:995::/var/lib/chrony:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
syslog:x:996:994::/home/syslog:/bin/false
lighthouse:x:1000:1000::/home/lighthouse:/bin/bash
tss:x:59:59:Account used by the trousers package to sandbox the tcsd
daemon:/dev/null:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
wangml:x:1001:1001::/home/wangml:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/false
zhangchao:x:1002:1002::/home/zhangchao:/bin/bash
(base) [root@CentOS-wangml shell]# awk -F ":" 'NR==1{print $1 $3 $5}'
^[[A^C
(base) [root@CentOS-wangml shell]# awk -F ":" 'NR==1{print $1 $3 $5}' passwd
root0root
(base) [root@CentOS-wangml shell]# awk -F ":" 'NR==1{print $1,$3,$5}' passwd
```

```
root 0 root
(base) [root@CentOS-wangml shell]# awk -F ":" 'NR==1{print $1 "-" $3 "-" $5}'
passwd
root-0-root
```

awk程序的优先级

1. BEGIN，在操作数据流之前执行，不需要数据源，可以直接执行
2. program，处理数据流
3. EBD，处理完数据流后再执行

```
(base) [root@CentOS-wangml shell]# awk 'BEGIN{print "awk process start..."}{print
$0}END{print "awk process stop..."}' test_sed.txt
awk process start...
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
awk process stop...
```

awk的高级用法

支持变量和数组

```
(base) [root@CentOS-wangml shell]# awk 'BEGIN{name="xiaobai";print name}'
xiaobai
(base) [root@CentOS-wangml shell]# awk 'BEGIN{arr[1]="xiaobai";arr[2]=18;print
arr[1], arr[2]}'
xiaobai 18
```

支持各类运算

1. 比较运算

```
(base) [root@CentOS-wangml shell]# cat num.txt
1
2
3
4
5
6
7
8
9
(base) [root@CentOS-wangml shell]# awk '$1==5{print $0}' num.txt
5
(base) [root@CentOS-wangml shell]# awk '$1<5{print $0}' num.txt
1
2
3
4
(base) [root@CentOS-wangml shell]# awk '$1>=5{print $0}' num.txt
5
6
7
```

```
8
9
```

2. 数学运算 + - * /
3. 逻辑运算 && | |
4. 精确(不)匹配 == !=
5. 模糊(不)匹配 ~ !~

流程控制

if else

```
(base) [root@CentOS-wangml shell]# awk '{if($1>5)print $0}' num.txt
6
7
8
9
(base) [root@CentOS-wangml shell]# awk '{
> if ($1<5)
> print $1*2
> else
> print $1/2
> }' num.txt
2
4
6
8
2.5
3
3.5
4
4.5
```

循环

> 1. for
> 2. while
> 3. break

```
(base) [root@CentOS-wangml shell]# awk -v 'sum=0' '{sum+=$1}END{print sum}'
num.txt
45
(base) [root@CentOS-wangml shell]# awk 'BEGIN{sum=0}{sum+=$1}END{print sum}'
num.txt
45
(base) [root@CentOS-wangml shell]# vim nums.txt
(base) [root@CentOS-wangml shell]# cat nums.txt
1 2 3 4 5
6 7 8 9 10
(base) [root@CentOS-wangml shell]# awk '{sum=0;for (i=1;i<6;i++){sum+=$i}print
sum}' nums.txt
15
40
(base) [root@CentOS-wangml shell]# awk '{sum=0;i=1;while(i<6){sum+=$i;i++}print
sum}' nums.txt
```

```
15
40
(base) [root@CentOS-wangml shell]# awk '{sum=0;i=1;while(i<6)
{sum+=$i;i++;if(sum>10){break}}print sum}' nums.txt
15
13
```

常用脚本示例

监控一个主机状态

```
(base) [root@CentOS-wangml shell]# cat ex1.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 监控目标主机状态

# 监控方法ping ICMP协议
# ping通 host up
# ping不通 host down

# 1. 关于禁止ping  防止DDOS
# 禁止的是陌生人

# 满足条件
# 网络延迟导致假报警问题
# ping的次数取值 3次全部失败则报警
# ping的评率 秒级 5秒 or 1秒

# main
for ((i=1;i<4;i++));do
    if ping -c1 $1 &>/dev/null;then
        export ping_count"$i"=1
    else
        export ping_count"$i"=0
    fi

    sleep 1
done

# 3次ping失败
if [ $ping_count1 -eq $ping_count2 ] && [ $ping_count2 -eq $ping_count3 ] && [
$ping_count1 -eq 0 ];then
    echo "$1 is down"
else
    echo "$1 is up"
fi

unset ping_count1
unset ping_count2
unset ping_count3

(base) [root@CentOS-wangml shell]# bash -x ex1.sh www.baidu.com
+ (( i=1 ))
+ (( i<4 ))
```

```
+ ping -c1 www.baidu.com
+ export ping_count1=1
+ ping_count1=1
+ sleep 1
+ (( i++ ))
+ (( i<4 ))
+ ping -c1 www.baidu.com
+ export ping_count2=1
+ ping_count2=1
+ sleep 1
+ (( i++ ))
+ (( i<4 ))
+ ping -c1 www.baidu.com
+ export ping_count3=1
+ ping_count3=1
+ sleep 1
+ (( i++ ))
+ (( i<4 ))
+ '[' 1 -eq 1 ']'
+ '[' 1 -eq 1 ']'
+ '[' 1 -eq 0 ']'
+ echo 'www.baidu.com is up'
www.baidu.com is up
+ unset ping_count1
+ unset ping_count2
+ unset ping_count3
```

监控一个端口状态

```
(base) [root@CentOS-wangml shell]# cat ex2.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 监控一个服务端口

# 监控方法
# 通过查看systemct1 service 服务启动状态
# lsof 查看端口是否存在
# 查看进程是否存在
####上述方式会出现问题：压力过大无法相应 或者 服务down了，但上述东西还在
# 测试端口是否有响应 推荐

port_status () {
# 创建一个临时文件
temp_file=`mktemp port_status.XXX`

# 判断依赖命令telnet是否存在
[ ! -x /usr/bin/telnet ] && echo "telnet: not found command"  && exit 1

# 测试端口 $1 IP $2 port
( telnet $1 $2 <<EOF
quit
EOF
) &>$temp_file

if egrep "\^]" $temp_file &>/dev/null;then
```

```
    echo "$1 $2 is open"
else
    echo "$1 $2 is down"
fi

rm -f $temp_file
}

port_status $1 $2


(base) [root@CentOS-wangml shell]# bash -x ex2.sh 127.0.0.1 22
+ port_status 127.0.0.1 22
++ mktemp port_status.XXX
+ temp_file=port_status.qDU
+ '[' '!' -x /usr/bin/telnet ']'
+ egrep '\^]' port_status.qDU
+ echo '127.0.0.1 22 is open'
127.0.0.1 22 is open
+ rm -f port_status.qDU
```

内存使用率

```
(base) [root@CentOS-wangml shell]# cat ex3.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 内存使用率统计脚本

# /proc/meminfo

# 内存申请顺序 free-cache-buffer-swap

memory_use() {
memory_used=`head -2 /proc/meminfo | awk 'NR==1{t=$2}NR==2{f=$2;print(t-
f)*100/t"%"}'`
memory_cache=`head -5 /proc/meminfo | awk 'NR==1{t=$2}NR==5{c=$2;print
c*100/t"%"}'`
memory_buffer=`head -4 /proc/meminfo | awk 'NR==1{t=$2}NR==4{b=$2;print
b*100/t"%"}'`

echo -e
"memory_used:$memory_used\tbuffer_used:$memory_buffer\tcached_used:$memory_cache
"
}

memory_use
(base) [root@CentOS-wangml shell]# bash ex3.sh
memory_used:96.1535%    buffer_used:0.108182%   cached_used:15.6095%
```

统计内存/CPU使用前十的进程

```
(base) [root@CentOS-wangml shell]# cat ex4.sh
#!/bin/bash

#Author: wangml
#Created Time:
#Script Description: 统计系统中前十名使用内存\CPU最多的进程

memory() {
    # 收集任务管理器进程信息
    temp_file=`mktemp memory.XXX`
    top -b -n 1 > $temp_file

    # 按进程统计内存使用大小
    tail -n +8 $temp_file | awk '{array[$NF]+=$6}END{for (i in array) print
array[i], i}' | sort -k 1 -n -r | head -10
    rm -f $temp_file
}

cpu() {
    # 收集任务管理器进程信息
    temp_file=`mktemp memory.XXX`
    top -b -n 1 > $temp_file

    # 按进程统计内存使用大小
    tail -n +8 $temp_file | awk '{array[$NF]+=$9}END{for (i in array) print
array[i], i}' | sort -k 1 -n -r | head -10
    rm -f $temp_file
}

echo "memory:"
memory
echo "cpu:"
cpu
(base) [root@CentOS-wangml shell]# bash ex4.sh
memory:
87100 go
25592 systemd-journal
23976 YDService
17712 barad_agent
13228 sshd
10120 iscsid
8916 bash
7544 rsyslogd
6788 BT-Task
4252 pickup
cpu:
6.2 barad_agent
0 YDService
0 YDLive
0 writeback
0 watchdogd
0 watchdog/0
0 vim
0 tuned
0 ttm_swap
```

```
0 top
```