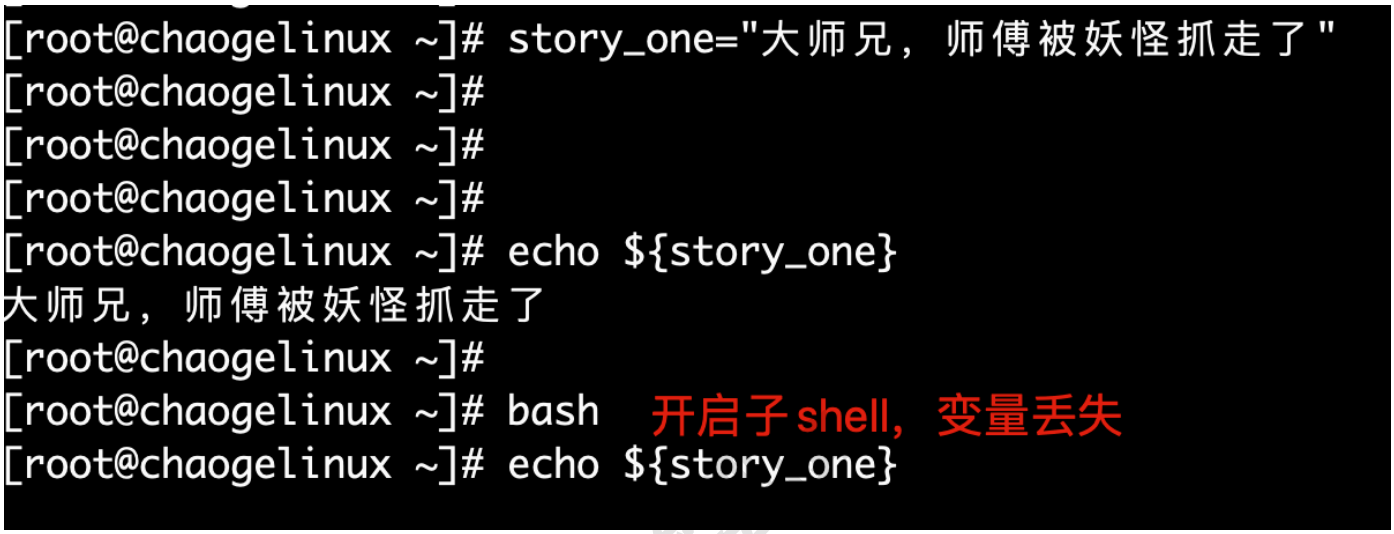
Shell变量

##本地变量

定义Shell变量，变量名不需要加美元符 `$`

本地变量只在用户当前shell生存期中有效，如



###变量定义

变量名要求：字母、数字、下划线组成、可以是 字母 或是 下划线 开头，如

- chaoge
- chao_ge123
- _chao_ge123

变量名严格区分大小写

- Chao_ge
- chao_ge

```
1.赋值不加引号
story_three=大师兄，快来救我

2.赋值单引号
story_two='大师兄，三师弟被妖怪抓走了'

3.赋值双引号
story_one="大师兄，师傅被妖怪抓走了"
```

取出变量值

- 单引号，所见即所得，强引用
- 双引号，输出引号里所有内容，识别特殊符号，弱引用
- 无引号，连续的符号可以不加引号，有空格则有歧义，最好使用双引号
- 反引号，引用命令执行结果，等于`$()`用法

特殊变量

shell的特殊变量，用在如脚本，函数传递参数使用，有如下特殊的，位置参数变量

```
$0    获取shell脚本文件名，以及脚本路径
$n    获取shell脚本的第n个参数,n在1~9之间，如$1 ,$2, $9 ，大于9则需要写，${10}，参数空格隔开
$#    获取执行的shell脚本后面的参数总个数
$*    获取shell脚本所有参数，不加引号等同于$@作用，加上引号"$*"作用是 接收所有参数为单个字符串，"$1 $2.."
$@    不加引号，效果同上，加引号，是接收所有参数为独立字符串，如"$1" "$2"  "$3" ...，空格保留
```

> 特殊变量实践

```
[root@chaogelinux tmp]# sh /tmp/p.sh yu chao cc
/tmp/p.sh yu chao

# 脚本内容
[root@chaogelinux tmp]# cat p.sh
#################################################################
# File Name: p.sh
# Author: pyyu
# mail: yc_uuu@163.com
# Created Time: 2020年05月25日 星期一 18时39分55秒
#================================================================
[root@chaogelinux shell_program]# cat special_var.sh
#! /bin/bash
echo '---特殊变量 $0 $1 $2 ..的实践'
echo '结果: '  $0 $1 $2

echo '######################'
echo '---特殊变量$# 获取参数总个数'
echo '结果: '  $#

echo '######################'
echo '---特殊变量$* 实践'
echo '结果: '  $*


echo '######################'

echo '---特殊变量$@ 实践'
echo '结果: ' $@
```

面试题分享

$*和$@ 的区别

$* 和 $@ 都表示传递给函数或脚本的所有参数

当 $* 和 $@ 不被双引号" "包围时，它们之间没有任何区别，都是将接收到的每个参数看做一份数据，彼此之间以空格来分隔。

但是当它们被双引号" "包含时，就会有区别了：
"$*"会将所有的参数从整体上看做一份数据，而不是把每个参数都看做一份数据。
"$@"仍然将每个参数都看作一份数据，彼此之间是独立的。

比如传递了 5 个参数，那么对于"$*"来说，这 5 个参数会合并到一起形成一份数据，它们之间是无法分割的；而对于"$@"来说，这 5 个参数是相互独立的，它们是 5 份数据。

如果使用 echo 直接输出"$*"和"$@"做对比，是看不出区别的；但如果使用 for 循环来逐个输出数据，立即就能看出区别来。

实践

```
[root@chaogelinux shell_program]# cat t1.sh
#!/bin/bash
echo "print each param from \"\$*\""
for var in "$*"
do
    echo "$var"
done
echo "print each param from \"\$@\""
for var in "$@"
do
    echo "$var"
done
```


特殊状态变量

```
$? 上一次命令执行状态返回值，0正确，非0失败
$$  当前shell脚本的进程号
$! 上一次后台进程的PID
$_ 再次之前执行的命令，最后一个参数

查找方式 man bash
  搜索Special Parameters
```

> 脚本控制返回值

```
[root@chaogelinux learnshell]# cat t4.sh
#################################################################
# File Name: t4.sh
# Author: pyyu
# mail: yc_uuu@163.com
# Created Time: 2020年05月26日 星期二 17时06分08秒
#================================================================
#!/bin/bash
[ $# -ne 2 ] && {
  echo "must be two args"
  exit 119 #终止程序运行，且返回119状态码，提供给当前shell的$?变量，若是在函数里 可以return 119用法
}
echo ok
```

上一次后台进程的PID

```
[root@chaogelinux shell_program]# nohup ping baidu.com & 1> /dev/null
[1] 21629
[root@chaogelinux shell_program]# nohup: 忽略输入并把输出追加到"nohup.out"

[root@chaogelinux shell_program]#
[root@chaogelinux shell_program]#
[root@chaogelinux shell_program]# echo $!
21629
[root@chaogelinux shell_program]# ps -ef|grep ping
root     21629 20999  0 15:46 pts/0    00:00:00 ping baidu.com
```


$$ 当前shell脚本的进程号

```
[root@chaogelinux shell_program]# cat special_var.sh
#! /bin/bash
echo '---特殊变量 $0 $1 $2 ..的实践'
echo '结果: '  $0 $1 $2

echo '######################'
echo '---特殊变量$# 获取参数总个数'
echo '结果: '  $#

echo '######################'
echo '---特殊变量$* 实践'
echo '结果: '  $*


echo '######################'

echo '---特殊变量$@ 实践'
echo '结果: ' $@

echo "当前脚本执行的进程号: $$"
```

> $_ 再次之前执行的命令，最后一个参数

```
[root@chaogelinux shell_program]# ls $_
special_var.sh

[root@chaogelinux shell_program]# cat $_
```

bash shell内置变量命令

bash本身提供的一些内置命令

```
echo
eval
exec
export
read
shift
```

echo命令

```
-n 不换行输出内容
-e 解析转义字符

\n  换行
\r  回车
\t  tab
\b  退格
\v  纵向制表符
```


案例

```
[root@chaogelinux learnshell]# echo chaoge;echo cc
chaoge
cc
[root@chaogelinux learnshell]# echo chaoge;echo cc -n
chaoge
cc -n
[root@chaogelinux learnshell]# echo -n  chaoge;echo cc
chaogecc
[root@chaogelinux learnshell]#
[root@chaogelinux learnshell]#
[root@chaogelinux learnshell]# echo "cc\tyy\tdd"
cc\tyy\tdd
[root@chaogelinux learnshell]# echo -e  "cc\tyy\tdd"
cc  yy  dd
[root@chaogelinux learnshell]# printf "cc\tyy\tdd\n"
cc  yy  dd
```

eval

```
eval 执行多个命令。
```

```
[root@chaogelinux shell_program]# eval ls ;cd /tmp
```

exec

```
不创建子进程，执行该命令，exec执行后自动exit

[root@chaogelinux learnshell]# exec date
2020年 05月 26日 星期二 17:28:03 CST
Connection to pyyuc closed.
```

shell子串

子串就是一个完整字符串的一部分，通过shell特有语法截取。

```
${变量}                      返回变量值
${#变量}                     返回变量长度，字符长度
${变量:start}                返回变量Offset数值之后的字符
${变量:start:length}     提取offset之后的length限制的字符
${变量#word}                 从变量开头删除最短匹配的word子串
${变量##word}                从变量开头，删除最长匹配的word
${变量%word}                 从变量结尾删除最短的word
${变量%%word}                从变量结尾开始删除最长匹配的word
${变量/pattern/string}   用string代替第一个匹配的pattern
${变量//pattern/string} 用string代替所有的pattern
```

案例

##子串基本用法

> Shell 截取字符串通常有两种方式：从指定位置开始截取和从指定字符（子字符串）开始截取。
>
> 从指定位置开始截取
>
> 这种方式需要两个参数：除了指定起始位置，还需要截取长度，才能最终确定要截取的字符串。
>
> 既然需要指定起始位置，那么就涉及到计数方向的问题，到底是从字符串左边开始计数，还是从字符串右边开始计数。答案是 Shell 同时支持两种计数方式。
>
> 1. 从字符串左边开始计数
>
> 如果想从字符串的左边开始计数，那么截取字符串的具体格式如下：
>
> ${string: start :length}
>
> 其中，string 是要截取的字符串，start 是起始位置（从左边开始，从 0 开始计数），length 是要截取的长度（省略的话表示直到字符串的末尾）。

```
[root@chaogelinux ~]# name="chao"
[root@chaogelinux ~]# echo ${name}
chao

[root@chaogelinux ~]# echo ${#name}
4

# 从start位置开始截取
[root@chaogelinux ~]# echo ${name:3}
o
[root@chaogelinux ~]# echo ${name:2}
ao
[root@chaogelinux ~]# echo ${name:1}
hao

# 指定start，以及元素长度
[root@chaogelinux ~]# echo ${name:1:2}
ha
```

> 计算变量值，长度的玩法


```
# 计算变量值，长度的玩法

[root@chaogelinux ~]# echo $name|wc -L #计算字符串长度
11
# 解释
# 打印行数
[root@chaogelinux shell_program]# cat test.txt |wc -l
2
# 打印最长行数的元素个数
[root@chaogelinux shell_program]# cat test.txt |wc -L
5

[root@chaogelinux ~]# expr length "$name"  #expr的length函数计算长度
11

[root@chaogelinux ~]# echo "$name" | awk '{print length($0)}' #用awk的length函数
11

#最快的方式
[root@chaogelinux ~]# echo ${#name}
11
```

> 字符串长度计算，多种方法，谁最快?

> 速度比较

```
# 最快方式
# seq -s 指定分隔符
# seq -s ":" 100
# 执行3次打印的命令，打印出一个指定了分隔符的1～100的序列
for n in {1..3};do char=`seq -s ":" 100`;echo ${char} ;done

# 实践
[root@chaogelinux ~]# time for n in {1..10000};do char=`seq -s "chaoge" 100`;echo ${#char}
&>/dev/null;done

real  0m11.041s
user  0m4.585s
sys 0m6.232s

#计算速度很慢，管道符和wc -L
[root@chaogelinux ~]# time for n in {1..10000};do char=`seq -s "chaoge" 100`;echo ${char}|wc -L
&>/dev/null;done

real  0m38.577s
user  0m15.394s
sys 0m22.491s

# 性能还不错
[root@chaogelinux ~]# time for n in {1..10000};do char=`seq -s "chaoge" 100`;expr length
"${char}" &>/dev/null;done

real  0m21.053s
user  0m8.673s
sys 0m11.944s

# awk再次加工，最慢
[root@chaogelinux ~]# time for n in {1..10000};do char=`seq -s "chaoge" 100`;echo ${char}|awk
'{print length($0)}' &>/dev/null ;done

real  0m33.728s
user  0m13.839s
sys 0m19.121s
```


shell编程，尽量用内置系统操作，与内置函数

截取字符串

> 基本语法

```
# 从开头删除匹配最短
## 从开头删除匹配最长
% 从结尾删除匹配最短
%% 从结尾删除匹配最长
```

```
# 指定字符内容截取
a*c  匹配开头为a，中间任意个字符，结尾为c的字符串
a*C  匹配开头为a，中间任意个字符，结尾为C的字符串



#语法
name="yuchao"  # 该变量的值，有索引，分别是从 0，1，2，3，4开始

${变量}                      返回变量值
${#name}                     返回变量长度，字符长度--------
${变量:start}                返回变量start数值之后的字符，且包含start的数字
${变量:start:length}         提取start之后的length限制的字符 ，例如${name:4:1}
${变量#word}                 从变量开头删除最短匹配的word子串 ${name:yu}
${变量##word}               从变量开头，删除最长匹配的word
${变量%word}                从变量结尾删除最短的word
${变量%%word}               从变量结尾开始删除最长匹配的word

替换
${变量/pattern/string}    用string代替第一个匹配的pattern
${变量//pattern/string}   用string代替所有的pattern
```


> 删除匹配的内容

```
[root@chaogelinux ~]# echo ${name}
I am chaoge

[root@chaogelinux ~]# echo ${name:2:2} #第二个开始，取2个
am

[root@chaogelinux ~]# name2=abcABC123ABCabc
[root@chaogelinux ~]#
[root@chaogelinux ~]
[root@chaogelinux ~]#

# 从开头删除
[root@chaogelinux ~]# echo ${name2#a*C}  #从开头删除最短的a*C
123ABCabc

[root@chaogelinux ~]# echo ${name2##a*C}  #从开头删除最长的匹配
abc

# 从结尾删除
# 从结尾没有匹配到结果，原样返回
[root@chaogelinux ~]# echo ${name2%a*C}
```

```
abcABC123ABCabc

# 匹配到了就删除
[root@chaogelinux ~]# echo ${name2%a*c}
abcABC123ABC

# 匹配长的删除

# 删干净了，因为变量值name2=abcABC123ABCabc，匹配a*c，取最长的也就从前删到结尾
[root@chaogelinux ~]# echo ${name2%%a*c}

# 原样返回，因为从结尾开始匹配，压根就找不到a*C，因此不做处理

[root@chaogelinux ~]# echo ${name2%%a*C}
abcABC123ABCabc
```

替换字符串

```
[root@chaogelinux ~]# str1="Hello,man,i am your brother."
[root@chaogelinux ~]#
[root@chaogelinux ~]#
[root@chaogelinux ~]#
[root@chaogelinux ~]# echo $str1
Hello,man,i am your brother.
[root@chaogelinux ~]#
[root@chaogelinux ~]#
# 一个/ 替换匹配第一个合适的字符串
[root@chaogelinux ~]# echo ${str1/brother/sister}
Hello,man,i am your sister.

# 两个//，匹配所有的合适的字符串
# 替换所有的o为大写O
[root@chaogelinux ~]# echo ${str1//o/O}
HellO,man,i am yOur brOther.
```


删除文件名练习

删除所有图片文件名中的子串

```
[root@chaogelinux ~]# touch stu_102999_{1..5}_finished.jpg
[root@chaogelinux ~]# touch stu_102999_{1..5}_finished.png
[root@chaogelinux ~]# ll *.jpg *.png
-rw-r--r-- 1 root root 0 5月  26 18:05 stu_102999_1_finished.jpg
-rw-r--r-- 1 root root 0 5月  26 18:07 stu_102999_1_finished.png
-rw-r--r-- 1 root root 0 5月  26 18:05 stu_102999_2_finished.jpg
-rw-r--r-- 1 root root 0 5月  26 18:07 stu_102999_2_finished.png
```

```
-rw-r--r-- 1 root root 0 5月  26 18:05 stu_102999_3_finished.jpg
-rw-r--r-- 1 root root 0 5月  26 18:07 stu_102999_3_finished.png
-rw-r--r-- 1 root root 0 5月  26 18:05 stu_102999_4_finished.jpg
-rw-r--r-- 1 root root 0 5月  26 18:07 stu_102999_4_finished.png
-rw-r--r-- 1 root root 0 5月  26 18:05 stu_102999_5_finished.jpg
-rw-r--r-- 1 root root 0 5月  26 18:07 stu_102999_5_finished.png
```

1.去掉所有 `_finished` 字符串

```
思路:
1.单个文件去掉后缀，很简单
[root@chaogelinux str1]# mv stu_102999_1_finished.jpg stu_102999_1.jpg

2.通过子串的替换方式
[root@chaogelinux str1]# f=stu_102999_1_finished.jpg
[root@chaogelinux str1]#
[root@chaogelinux str1]#
# 变量的子串功能，去掉后缀
[root@chaogelinux str1]# echo ${f//_finished/}
stu_102999_1.jpg

# 利用变量的反引用替换文件名
[root@chaogelinux str1]# mv $f `echo ${f//_finished/}`

# 剩下的文件，利用循环操作
# 找出剩下所有需要替换的jpg文件
[root@chaogelinux str1]# ls *fin*.jpg
stu_102999_2_finished.jpg  stu_102999_3_finished.jpg  stu_102999_4_finished.jpg
 stu_102999_5_finished.jpg
[root@chaogelinux str1]#

# 写shell循环代码，循环操作
# 去掉所有jpg文件的_finished后缀
[root@chaogelinux str1]# for file in `ls *fin*.jpg`;do mv $file `echo ${file//_finished/}`;done
[root@chaogelinux str1]# ls *.jpg
stu_102999_1.jpg  stu_102999_2.jpg  stu_102999_3.jpg  stu_102999_4.jpg  stu_102999_5.jpg
[root@chaogelinux str1]#
```


特殊shell扩展变量处理

语法

parameter，参数，范围

```
如果parameter变量值为空，返回word字符串
${parameter:-word}


如果para变量为空，则word替代变量值，且返回其值
${parameter:=word}
```

```
如果para变量为空，word当作stderr输出，否则输出变量值
用于设置变量为空导致错误时，返回的错误信息
${parameter:?word}


如果para变量为空，什么都不做，否则word返回
${parameter:+word}
```

> 扩展变量实践

> 演示1

:-

```
[root@chaogelinux str1]# echo $chaoge

[root@chaogelinux str1]#
[root@chaogelinux str1]#
# 当chaoge没有值，heihei被返回，赋值给result
[root@chaogelinux str1]# result=${chaoge:-heihei}
[root@chaogelinux str1]#
[root@chaogelinux str1]# echo $result
heihei
# 要注意的是，此时chaoge还是空
[root@chaogelinux str1]# echo $chaoge

[root@chaogelinux str1]#


# 情况2，当chaoge变量有值时，该特殊扩展变量的符号，也就不起作用了
[root@chaogelinux str1]# echo $chaoge
pangzi

[root@chaogelinux str1]#
[root@chaogelinux str1]# result=${chaoge:-heihei}
[root@chaogelinux str1]# echo $result
pangzi
[root@chaogelinux str1]# echo $chaoge
pangzi
```


> 演示2

:=

该特殊情况用于保证变量始终有值

```
# 撤销变量
[root@chaogelinux str1]# echo $chaoge
[root@chaogelinux str1]# unset result
```

```
# 发现，hehe不但给了result，还给了chaoge变量
[root@chaogelinux str1]# result=${chaoge:=hehe}
[root@chaogelinux str1]# echo $result
hehe
[root@chaogelinux str1]# echo $chaoge
hehe

# 如果变量有值，什么事也不做
[root@chaogelinux str1]# result=${chaoge:apple}
[root@chaogelinux str1]# echo $result
hehe
[root@chaogelinux str1]# echo $chaoge
hehe
```

> 演示3
>
> :?，当变量不存在时候，输出指定信息

```
[root@chaogelinux str1]# echo ${cc}

# 默认错误

[root@chaogelinux str1]# echo ${cc:?}
-bash: cc: 参数为空或未设置

[root@chaogelinux str1]# echo ${cc:?cc不存在}
-bash: cc: cc不存在

# 变量有值，则不做处理
[root@chaogelinux str1]# cc="happy"
[root@chaogelinux str1]# echo ${cc:?cc不存在}
happy
```


> 演示4
>
> :+ 如果变量为空，什么都不做，否则替换

```
[root@chaogelinux str1]# unset cc result chaoge

# 为空
[root@chaogelinux ~]# result=${name:+chaoge}
[root@chaogelinux ~]# echo $result

[root@chaogelinux ~]# echo $name

[root@chaogelinux ~]#

# 不为空
[root@chaogelinux ~]# name="xiaoyu"
[root@chaogelinux ~]#
# 后面的值，返回给result
```

```
[root@chaogelinux ~]# result=${name:+chaoge}
[root@chaogelinux ~]# echo $result
chaoge
[root@chaogelinux ~]# echo $name
xiaoyu
```

扩展变量的应用场景

在脚本开发中，例如数据备份、删除的脚本

> 删除7天前的过期数据

```
[root@chaogelinux shell_program]# cat del_data.sh
find ${path:=/tmp} -name '*.tar.gz' -type f -mtime +7|xargs rm -f

# 上述就对path变量做了处理，否则如果path变量为定义，命令就会报错
# 有误的脚本，未指定path的路径，就会在当前目录删除，程序就有了歧义，bug
[root@chaogelinux shell_program]# cat del_data.sh
find ${path} -name '*.tar.gz' -type f -mtime +7|xargs rm -f
[root@chaogelinux shell_program]#
```