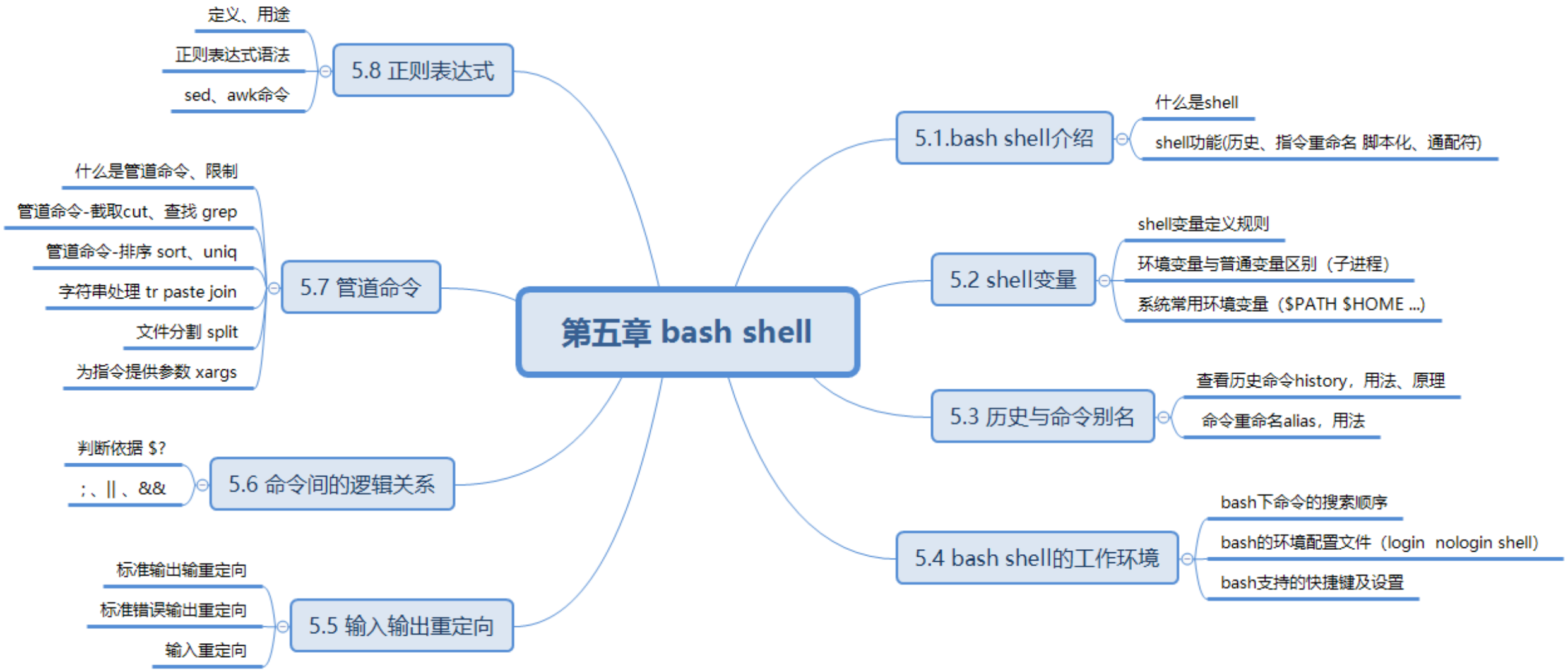

第五章 bash shell
5.1.bash shell介绍
什么是shell
shell功能(历史、指令重命名 脚本化、通配符)
5.2 shell变量
shell变量定义规则
环境变量与普通变量区别（子进程）
系统常用环境变量（$PATH $HOME ...）
5.3 历史与命令别名
查看历史命令history，用法、原理
命令重命名alias，用法
5.4 bash shell的工作环境
bash下命令的搜索顺序
bash的环境配置文件（login nologin shell）
bash支持的快捷键及设置
5.5 输入输出重定向
标准输出输重定向
标准错误输出重定向
输入重定向
5.6 命令间的逻辑关系
判断依据 $?
;、||、&&
5.7 管道命令
什么是管道命令、限制
管道命令-截取cut、查找 grep
管道命令-排序 sort、uniq
字符串处理 tr paste join
文件分割 split
为指令提供参数 xargs
5.8 正则表达式
定义、用途
正则表达式语法
sed、awk命令

# 第6章

# shell script

# 第6章 shell 脚本

- 6.1 什么是shell脚本
- 6.2 shell脚本的执行方式
- 6.3 shell脚本的条件判断
- 6.4 shell脚本的分支结构
- 6.5 shell脚本的循环结构
- 6.6 shell脚本的调试

# 6.1 什么是shell脚本

# 什么是shell脚本

- Shell的作用是解释执行用户的命令，用户输入一条命令，Shell就解释执行一条，这种方式称为交互式（Interactive），Shell还有一种执行命令的方式称为批处理，用户事先写一个Shell脚本（Script），其中有很多条命令，让Shell一次把这些命令执行完，而不必一条一条地敲命令。

# 什么是shell脚本

- shell脚本是以行为单位执行的，在执行脚本的时候会分解成一行一行依次执行。脚本通常以sh为扩展名，包含的成分主要有注释、命令、Shell变量和流程控制语句。其中：

① 注释。用于对脚本进行解释和说明，在注释行的前要加上“#”

② 命令。在Shell脚本中可以出现任何在交互方式下使用的命令。

③ Shell变量。Shell支持具有字符串值的变量。

④ 流程控制。主要为一些用于流程控制的内部命令。

# 什么是shell脚本

- Shell脚本用途

  自动化管理的重要依据（硬件状态、流量追踪、软件更新）

  追踪与管理系统重要工作

  连续指令单一化

  简单的数据处理

  跨平台支持且易学习

  缺点：执行效率低

# 什么是shell脚本

- 第一个shell脚本

  脚本组成部分：

  1、#!/bin/bash 声明脚本使用的shell名称

  2、程序内容的说明

  3、脚本内容

  4、定义返回值

```
#!/bin/bash

#Fuction:
#       show "hello world" on the screen
#Time: 2020/08/20
#Author:liu

echo "Hello World!"

echo -e "hello China! \n"

exit 0
```

# 什么是shell脚本

- shell脚本说明

1. 脚本文件是纯文本；
2. 分析运行：从上到下，从左到右；
3. 命令与选项间的多个空白会被忽略；
4. 空白行会被忽略，tab视作空格；
5. 如果读到『回车』，则尝试运行；
6. 一行内容过多，可以用『\』来延伸至下一行；
7. #为注释，其后的数据会被忽略；

# 什么是shell脚本

- 编写shell脚本的好习惯

脚本的说明(作者、时间、版本、版权、历史…);

注释;

缩进;

最好使用vim;

# shell脚本举例

1. 交互式脚本

read命令可以从键盘读入，通过read命令可以实现从键盘输入值的交互式脚本。

例：用键盘输入为firstname， lastname两个变量赋值，打印出全名。

```
#!/bin/bash

#Funcation:
#
#Time:
#Version:
#Author:liu

read -p "Pleas input your first name:" firstname

read -p "Input your second name:" secondname

echo "Your Name is: $firstname $secondname"
```

# shell脚本举例

2. 创建文件名随日期变化的文件

备份需要每天进行，为了避免重复，就需要用当天的日期来对文件进行命名。

```
# input filename
read -p "Please input your filename:" fileuser

date1=$(date +%Y%m%d)
file1=${fileuser}_${date1}

date2=$(date --date='2 days ago' +%Y%m%d)
file2=${fileuser}_${date2}

touch "${file1}"
touch ${file3}
```

注：指令内容赋值方式$()，``；

变量的拼接

# shell脚本举例

3. 数值运算

例：输入两个数，计算和与乘积

```
read -p "Pleas input a Number:" a
read -p "Input another number:" b

#total=$a+$b

declare -i total=$a+$b

echo "The sum of $a + $b is:" $total

mul=$(($a*$b))
echo "The mul of $a * $b is:" $mul
~
```

注：bash shell预设仅支持整数，需要用declare -i 将变量声明为整数才能运算。此外，还可使用$((计算表达式))来进行数值运算。

# 6.2 shell script的执行方式

# shell脚本的执行方式

- 1.直接指令下达（以first.sh为例）

  要求：first.sh 文件必须具备可读与可执行的权限

  执行方式：

  绝对路径：使用/home/liu/first.sh 下达指令

  相对路径：当前目录/home/liu, 使用./first.sh

  将first.sh放到环境变量PATH指定的目录中，/home/liu/bin可以尝试

# shell脚本的执行方式

- 2.以bash程序来执行

  要求：first.sh 文件只需要有读权限

  执行方式：

  bash first.sh

  sh first.sh

  sh 参数：

  sh -n 只读shell脚本，检查语法，不执行

  sh -x 显示执行的每条指令

# shell脚本的执行方式

- 3.source命令

  要求：first.sh 文件只需要有读权限

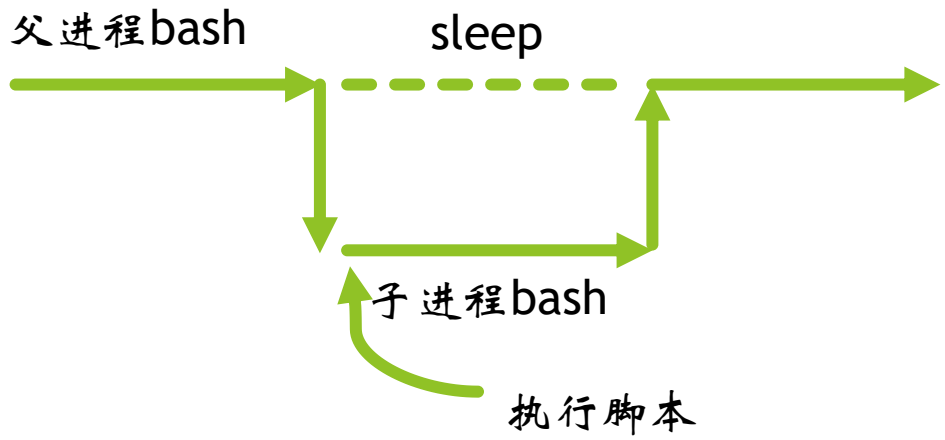
  执行方式：

  source first.sh

# shell脚本的执行方式

- 脚本执行方式的差别

直接指令下达、sh命令：执行时会调用新的bash（子进程）来执行脚本中的指令。执行完成后，回到父进程，子进程中的变量和动作结束后不会回传给父进程。



# shell脚本的执行方式

- 3种脚本执行方式的差别

source命令执行：会直接在父进程执行，因此各项变量、动作都会在原bash中生效。

# 6.3 shell 脚本的条件判断

# shell脚本的判断式

- 写脚本时，有时需要判断字符串是否相等，检查文件状态或者数字测试，基于这些测试才能进行下一步的操作。
- shell脚本编程使用“test”命令和“[]”符号实现判断。

复习：根据之前学习的内容，如何实现判断文件是否存在，如果不存在则创建文件。

提示：&&　　||　　$?

# shell脚本的判断式

- 1.test命令

  用法：test [-选项] 文件\数值\字符串

  说明：执行结果没有任何输出，同样可以用$?、&&、||显示结果

  选项：

  test命令选项丰富，可以用来判断：

  文件类型

  文件权限

  数值关系

  字符串

  …..

# shell脚本的判断式

- 文件名、文件类型判断

| 测试标志 | 代表意义 |
| --- | --- |
| -e | 该文件名是否存在 |
| -f | 该文件名是否存在且为file |
| -d | 该文件名是否存在且为目录 |
| -b | 该文件名是否存在且为一个block |
| -c | 该文件名是否存在且为一个character device设备 |
| -S | 该文件名是否存在且为一个socket文件 |
| -p | 该文件名是否存在且为一个FIFO（pipe）文件 |
| -L | 该文件名是否存在且为一个连接文件 |

test –e file1

$?  == 0

test –r file1

# shell脚本的判断式

- 文件权限判断

| 测试标志 | 代表意义 |
| --- | --- |
| -r | 检测文件名是否存在且具有“可读”权限 |
| -w | 检测文件名是否存在且具有“可写”权限 |
| -x | 检测文件名是否存在且具有“可执行”权限 |
| -u | 检测文件名是否存在且具有“SUID”权限 |
| -g | 检测文件名是否存在且具有“SGID”权限 |
| -k | 检测文件名是否存在且具有“Sticky bit”权限 |
| -s | 检测文件名是否存在且为“非空白文件” |

# shell脚本的判断式

- 文件比较

| 测试标志 | 代表意义 |
| --- | --- |
| -nt | （newer than）判断file1是否比file2新 |
| -ot | （older than）判断file是否比file2旧 |
| -ef | 判断file1与file2是否为同一个文件，可用在判断hard link上 |

# shell脚本的判断式

- 数值判断

| 测试标志 | 代表意义 |
| --- | --- |
| -eq | equal |
| -ne | not equal |
| -gt | greater than |
| -lt | less than |
| -ge | greater than or equal |
| -le | less than or equal |

# shell脚本的判断式

- 字符串判断

| | |
|---|---|
| -z string | 判断字符串是否为空，若为空返回true |
| -n string | 判断字符串是否为非空，若为非空返回true |
| test str1 == str2 | 判断str1是否等于str2，若相等，返回true |
| test str1 != str2 | 判断str1是否等于str2，若相等，返回false |

# shell脚本的判断式

- 多重条件判断

| 测试标志 | 代表意义 |
| --- | --- |
| -a | 两个条件同时成立，eg：test -r file -a test -x file，同时成立返回true |
| -o | 任何一个条件成立，eg：test -r file -o test -x file，同时成立返回true |
| ！ | 反向状态，如test！-x file，当file不具有x时，返回true |

# shell脚本的判断式

- test判断命

要求：编 存在当前文件夹下

不

存 出文件类型。判断当前用户对该文件所拥有的权限。

```
#Author:liu

read -p "please input a file name:" filename

# 判断输入是否为空
test -z $filename && echo "Please input a filename." && exit 0

test ! -e ${filename} && echo "File ${filename} not exsit" && exit 0

test -f ${filename} && filetype="regular file"
test -d ${filename} && filetype="directory"
test -r ${filename} && perm="readable"
test -w ${filename} && perm="${perm} writable"
test -x ${filename} && perm="${perm} executable"

echo "${filename} is a ${filetype}"
echo "$And the permission for you are ${perm}"
```

# shell脚本的判断式

- 2.[]符号

  用法：[ -选项　文件/字符 ]

  说明：中括号判断方式，测试标志和test相同。

  使用‘[]’测试时需要注意：

  中括号两端要有空格，中括号内的每个选项\符号间都需要有空格。

  中括号内的变量，最好用双引号括起来，避免出问题

  中括号内的常量，最好用单引号或双引号括起来；

  例：[“${name}”==“abc” ]　　[ “${name}” == “abc” ]

# shell脚本的判断式

- 举例

要求：用户选择输入y/Y或n/N，根据用户输入显示不同的信息。如果用户输入的不是y/Y，n/N则输出选择不合适。

```
read -p "Please input (Y/N):" yn

[ "${yn}" == "Y" -o "${yn}" == "y" ] && echo "ok, continue" && exit 0
[ "${yn}" == "N" -o "${yn}" == "n" ] && echo "no,interrupt" && exit 0

echo "Your choice is not reasonable!"
~
```

# shell脚本传参

- shell脚本和指令一样，执行时也可以带参数。

  shell脚本的参数有固定的名称，在脚本中使用这些名称应用变量：

  $0：执行的脚本名称

  $1、$2..：脚本携带的第一个参数、第二个参数..

  $#：后面接的变量的个数

  $@：代表“$1” “$2”变量列表，每个变量是独立的，用双引号括起来

  $*：代表“$1 $2 $3”，默认用c对变量进行分隔。

# shell脚本传参

- 参数使用举例

  编写携带参数的脚本，执行后屏幕显示：

  程序的文件名

  共有几个参数

  若参数的个数小于2则告只使用者参数太少

  全部参数内容如何

  第一个参数是什么

  第二个参数是什么

```
echo "The script name is: ${0}"

echo "Total parameter number is: $#"

[ $# -lt 2 ] && echo "The number of parameter is less than 2!" && exit 0

echo "Parameter is: $@ "

echo "The 1st paramter is: ${1}"
echo "The 2nd paramter is: ${2}"
~
```

# 6.4 分支结构

# 分支结构

- 单层条件判断

```
if [ 条件判断式 ]; then
    成立时，执行
fi
```

条件判断式可以用test 或 [] ,当多个条件时可以用 –a –o, 也可两个条件间用&& || 分隔。

```
[  “${yn}” ==“Y” -o “${yn}” ==“y” ]
[  “${yn}” ==“Y” ]  || [ “${yn}” ==“y”  ]
```

# 分支结构

- 多重条件判断

```
if [ 条件判断式 ]; then
        符合条件，执行
else
        条件判断不成立时，执行
fi
```

# 分支结构

- 多重条件判断

```
if  [ 条件判断式1 ]; then
        符合条件判断1，执行
elif [ 条件判断式2 ]; then
        复合条件判断2，执行
else
        当判断式1和2都不成立时，执行
fi
```

# 分支结构

- 多重条件判断(举例）

编写一个shell脚本，输入（1-10）之间的一个数，并判断它是否小于5。

```
echo 'key in a number (1 - 10 ) :
read a
if [ “$a” -lt 1 –o “$a” -gt 10 ];then
        echo  " Error Number . "
        exit
elif [ ! "$a" –lt 5 ];then
        echo  " It's not less 5 . "
else
        echo  " It's less 5 . "
fi
```

# 分支结构

- case…esac 判断

对同一变量进行多次的测试，比elif语句更简单、简洁，变量逐个和str1，str2比较，相等则执行command-list1， 若没有相等的字符串，则执行*后的语句。

```
case $变量 in
“str1”)
    commands-list1;;
“str2”)
    commands-list2;;
                ...
*)
    commands-listn;;
    esac
```

# 分支结构

- case…esac 举例

```
case ${1} in
"hello")
        echo "hello ?";;
"nihao")
        echo "nihao !";;
*)
        echo "others";;
esac
```

# 6.5 shell script中的循环结构

# 循环结构

- while do done

当判断式条件成立时执行

```
while [ 判断式 ]
do          # 循环开始
      执行程序
done       # 循环结束
```

# 循环结构

- while do done（举例）

编写程序，这段程序对各个给定的位置参数，首先判断其是否是普通文件，若是，则显示其内容；否则，显示它不是文件名的信息。

```
while [ "${yn}" != "yes" -a "${yn}" != "YES" ]
do
        read -p "please input yes/Yes to stop this program:" yn
done
echo "OK! you input a yes."

read -p "please input a number" num
i=1
while [ ${i} -le "${num}" ]
do
        sum=$((${sum}+${i}))
        i=$((${i}+1))
done
echo "sum is ${sum}"
```

# 循环结构

- util do done

当判断式条件不成立时执行

until [ 条件判断 ]

do

　　程序段落

done

# 循环结构

- util do done（举例）

将while中的例子改为用until实现

# 循环结构

- for…do…done(固定循环）

与while需判断条件不同，for循环已知循环次数

for var in con1 con2 con3 …

do

　　程序段

done

# 循环结构

- for…do…done的数值处理

for (( 初始值；限制值；执行步长 ))

do

　　程序段

done

　　初始值：变量开始值，如i=1

　　限制值：当变量在这个限制范围内，就继续循环，如i<=100

　　执行步长：每次循环的变化量，如i=i+1

# 循环结构

- 循环控制break

break语句用于从for、while、until循环中退出，停止循环的执行。

break语句的语法如下所示：

break [n]

n代表嵌套循环的层级，如果指定了n，break将退出n级嵌套循环。默认n=1如果没有指定n或n不大于等于1，则退出状态码为0，否则退出状态码为n。

# 循环结构

- 循环控制continue

continue语句用于跳过循环体中剩余的命令直接跳转到循环体的顶部，而重新开始循环的下一次重复。continue语句可以应用于for、while或until循环。continue语句的语法如下所示：

continue [n]：

把n层循环剩余的代码都去掉，但是循环的次数不变。默认n=1。

# 循环结构

- break和continue对比

```
#!/bin/sh
for i in a b c d
 do
echo -n $i
   for j in 1 2 3 4 5 6 7 8 9 10
    do
      if [ $j -eq 5 ];then
        break或continue
      fi
      echo -n “ $j”
    done
echo
 done
```

break结果：
a 1 2 3 4
b 1 2 3 4
c 1 2 3 4
d 1 2 3 4

break 2的结果：
a 1 2 3 4

continue结果：
a 1 2 3 4 6 7 8 9 10
b 1 2 3 4 6 7 8 9 10
c 1 2 3 4 6 7 8 9 10
d 1 2 3 4 6 7 8 9 10

continue 2的结果：
a 1 2 3 4b 1 2 3 4c 1 2 3 4d 1 2 3 4

# 6.6 shell script中的函数

# 函数

- shell脚本中的函数

shell 脚本同样支持函数，可以减少重复执行的代码段，简化程序代码。

注意：shell脚本中的函数一定要在脚本的最前面定义。

# 函数

- 定义函数的语法：

function_name()

{ # 函数体，在函数中执行的命令行

commands…

# 参数返回，return语句是可选的。如果没有return语句，则以函数最后一条命令的运行结果作为返回值；如果使用return语句，则return后跟数值n（数值范围：0~255）

[ return int; ]

}

# 函数

- 定义函数的语法：

也可在函数名前加上关键字function。

```
function function_name()
{
  commands…
 }
```

如果有function关键字，则可以省略圆括号“()”。函数体，也叫复合命令块，是包含在{}之间的命令列表。

# 函数

- 定义函数的语法：

可以在一行内定义一个函数，此时，函数体内的各命令之间必须用分号“；”隔开，其语法规则如下：

function name { command1; command2; commandN; } # 注意 大括号前后需空格

或者

name() { command1; command2; commandN; }

# 函数

- 函数的调用

语法：函数名 参数1 参数2 … 参数n

例：name foo bar

name = 函数名

foo = 参数1：传递给函数的第一个参数（位置参数$1）

bar = 参数2：传递给函数的第二个参数（位置参数$2）

```
#!/bin/bash
function show() {
    echo "hello , you are calling the function  $1"
}
echo "first time call the function"
show first
echo "second time call the function"
show second
```

# 函数

- shell脚本参数与函数参数

  传：shell脚本传参:  ./first.sh  a b c

  函数传参: fun a b c

  取：取shell脚本的参数，$0, $1,$2…

  取函数的参数，$0, $1,$2…

虽然二者传递、取用参数的方式相似，但较容易区分。函数中取用的$0 $1…一定是调用函数时传递的参数，而shell脚本中的$0 $1来自与脚本执行时传递的参数。

```
function name {
 echo “your name is ${1}"
}
name zhang
echo ${1}
```

# 函数

- 本地变量

默认条件下，在函数和shell主体中建立的变量都是全局变量，可以相互引用，当shell主体部分与函数部分拥有名字相同的变量时，可能会相互影响

```
[~/shell/function]# ./variable.sh
12 3 2
temp is: 5
value is: 6
the result is 72
temp is larger
```

```
#!/bin/bash
if [ $# -ne 3 ]
then
    echo "usage: $0 a b c"
    exit
fi
temp=5
value=6
echo temp is: $temp
echo value is: $value
fun3() {
    temp=`echo $(($1*$2*$3))`
    result=$temp
}
fun3 $1 $2 $3
echo "the result is $result"
if [ $temp -gt $value ]
then
    echo "temp is larger"
else
    echo "temp is still smaller"
fi
```

# 函数

- 本地变量

使用local命令来创建一个本地变量，其语法：

```
local var=value
local varName
function name(){
    # 定义一个本地变量var
    local var=$1
    command  on $var
}
```

local命令只能在函数内部使用。

local命令将变量名的可见范围限制在函数内部。

使用局部变量，使得函数在执行完毕后，自动释放变量所占用的内存空间，从而减少系统资源的消耗，在运行大型的程序时，定义和使用局部变量尤为重要。

# 函数

```
#!/bin/bash
if [ $# -ne 3 ]
then
    echo "usage: $0 a b c"
    exit
fi
temp=5
value=6
echo temp is: $temp
echo value is: $value
fun3() {
    (local) temp=`echo $(($1*$2*$3))`
    result=$temp
}
fun3 $1 $2 $3
echo "the result is $result"
if [ $temp -gt $value ]
then
    echo "temp is larger"
else
    echo "temp is still smaller"
fi
```

```
[~/shell/function]# ./variable.sh
12 3 2
temp is: 5
value is: 6
the result is 72
temp is larger
```

```
[~/shell/function]# ./variable.sh
12 3 2
temp is: 5
value is: 6
the result is 72
temp is still smaller
```

# 函数

- 函数返回值

如果在函数里有shell内置命令return，则函数执行到return语句结束，并且返回到shell脚本中调用函数位置的下一个命令。如果return带有一个数值型参数，则这个参数就是函数的返回值，返回值的最大值是255；否则，函数的返回值是函数体内最后一个执行的命令的返回状态。

函数中的关键字“return”可以放到函数体的任意位置，通常用于返回某些值，shell在执行到return之后，就停止往下执行，返回到主程序的调用行，return的返回值只能是0~255之间的一个整数，返回值将保存到变量“$?”中。

# 小结

函数定义：function function_name 、 function_name()

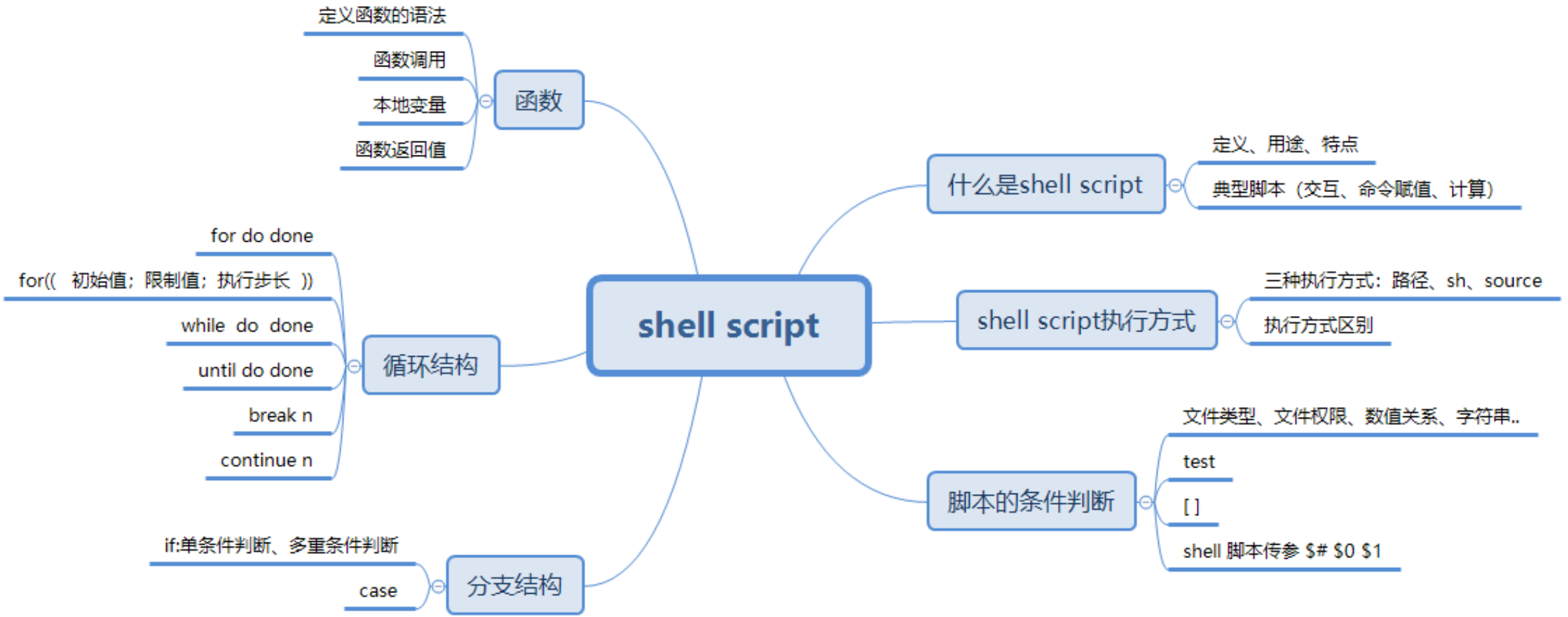
函数调用：function_name  a b c

函数参数：$0 、$1…

本地变量：函数内定义，作用域在函数内

函数返回值：0~255、$?取用

# 章小结


shell script
函数
定义函数的语法
函数调用
本地变量
函数返回值
循环结构
for do done
for(( 初始值；限制值；执行步长 ))
while do done
until do done
break n
continue n
分支结构
if:单条件判断、多重条件判断
case
什么是shell script
定义、用途、特点
典型脚本（交互、命令赋值、计算）
shell script执行方式
三种执行方式：路径、sh、source
执行方式区别
脚本的条件判断
文件类型、文件权限、数值关系、字符串..
test
[ ]
shell 脚本传参 $# $0 $1