shell学习心得

shebang 指的是出现在文本文件的第一行前两个字符

#!/bin/sh 通过 sh 去执行

#!/usr/bin/python 通过 Python 去执行

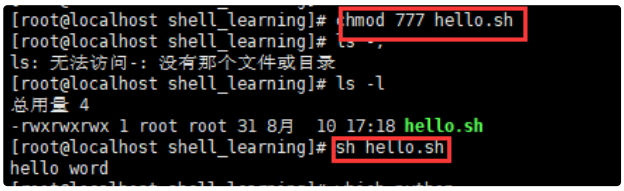
#1/usr/bin/env 在不同平台上都能正确找到解释器的办法

在未指定解释器的时候 会默认使用 sh 解释器去执行文件

#代表 1 个注释符 在#后的文件不生效

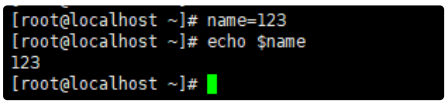
新建的 shell 文件需要用 chmod 赋予执行权限 不然无法执行

如果觉得权限过于高的话 尽量选择 chmod + x



变量理解学习

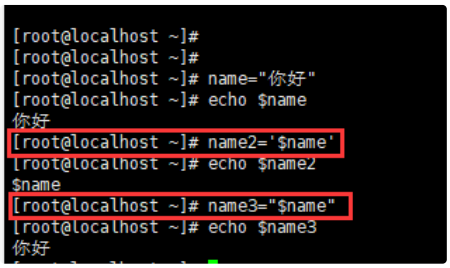
变量是暂时存储数据的地方,是一种数据标记,通过调用正确的变量名字,可取出对应的值,字符串的变量 需要加引号



变量名规则 不可使用保留关键字,不能以数字开头 不能用标点符号

变量名区分大小写

> 单引号变量,不识别特殊语法 双引号变量,能识别特殊符号



shell 的特殊变量,用在脚本,函数传递参数使用

用不同的方式,执行脚本,产生的后果也不一样

source 和. 是在当前的 shell 环境中加载变量,执行脚本

bash 或 sh 是在子 shell 运行的,变量也在子 shell 中加载运行

```
[root@localhost shell_learning]# echo $name

[root@localhost shell_learning]# echo $day

[root@localhost shell_learning]# source make_var.sh
chen 已经学习了shell 15天
[root@localhost shell_learning]# echo $name
chen
[root@localhost shell_learning]# echo $day
15
[root@localhost shell_learning]#
```

$0 ———– 当前脚本的文件名（不带路经，带扩展名 .sh ）

$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

$# ———– 统计传入的变量个数

$* ———– 列出所有参数 。” $* ” 整体列出

$@———– 列出所有参数 ,加了引号以后 接收所有参数为独立字符串

参数的理解

bash test1.sh 参数 1 参数 2 参数 3

实践脚本

```
#! /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$@实践'
echo '结果:'$@
~
```

```
[root@localhost shell_learning]# bash special_var.sh chen chen chen chen
特殊变量 $0,$1,$2的实践
结果: special_var.sh chen chen
######################
特殊变量$# 获取参数总个数
结果: 4
######################
特殊变量$* 实践
结果:chen chen chen chen
######################
特殊变量$@实践
结果:chen chen chen chen
[root@localhost shell_learning]#
```

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 ID

```
[root@localhost shell_learning]# ls aa
ls: 无法访问aa: 没有那个文件或目录
[root@localhost shell_learning]# echo $?
2
[root@localhost shell_learning]# ls
hello.sh  special_var.sh
[root@localhost shell_learning]# echo $?
0
```

只有执行正确命令的时候 $?返回正常值 0 反之为 0

脚本返回值,执行完以后,会返回数字 ID 称之为返回值

```
#!/bin/bash
#收集参数,如果参数不等于1 则显示文字超过2个参数
[ $# -ne 1 ] && {
   echo '2个参数及以上'
   exit '119' #终止程序运行
}
echo '1个参数'
~
```

```
[root@localhost shell_learning]# bash teshu.sh 180 180
2个参数及以上
[root@localhost shell_learning]# echo $?
119
[root@localhost shell_learning]# echo $?
0
```

$!返回上次后台进程的 ID

```
[root@localhost dev]# nohup ping www.baidu.com & 1> /dev/null
[2] 9406
[root@localhost dev]# nohup: 忽略输入并把输出追加到"nohup.out"

[root@localhost dev]# ps -ef | grep ping
root       9394   7452  0 11:44 pts/0    00:00:00 ping www.baidu.com
root       9406   7452  0 11:44 pts/0    00:00:00 ping www.baidu.com
root       9434   7452  0 11:45 pts/0    00:00:00 grep --color=auto ping
[root@localhost dev]# enho $!
bash: enho: 未找到命令...
[root@localhost dev]# echo $!
9406
```

内置命令、外置命令学习,以及其他学习

内置命令:在系统启动时加载内存、常驻内存、执行率高， 占用资源

外置命令:用户需要从硬盘中读取的文件,再进行加载

```
[root@localhost ~]# type cd
cd 是 shell 内嵌
[root@localhost ~]# type bash
bash 是 /usr/bin/bash
[root@localhost ~]#
```

cd 就是 shell 内嵌 就是内置命令

外置命令也可以算是 单独下载的文件系统命令,例如 tree,nginx,tomcat

可以通过 linux 的 type 的命令,验证是否是内置,或外置命令

同时 shell 支持执行多行命令

```
[root@localhost ~]# date;whoami;ls;cd /usr
2022年 08月 15日 星期一 10:33:13 CST
root
anaconda-ks.cfg  lovers.txt  pwd2.txt      Python-3.7.1.tgz  shuaige.txt  wudi      公共  图片  音乐
dog.txt          nohup.out   pwd.txt       ruanlianjie       test.txt     www       模板  文档  桌面
hello.sh         opt         Python-3.7.1  shell             tlink        yum.conf  视频  下载
[root@localhost usr]#
```

echo 命令,可以在 linux 下进行格式化打印

```
#!/bin/bash

#这里用于显示日期,和谁登陆
echo "当前系统时间是"
date

echo "谁登陆了系统"
whoami
~
```

```
[root@localhost shell_learning]# cat echo_test.sh
#!/bin/bash

#这里用于显示日期,和谁登陆
echo "当前系统时间是"
date

echo "谁登陆了系统"
whoami
[root@localhost shell_learning]# sh echo_test.sh
当前系统时间是
2022年 08月 15日 星期一 10:45:37 CST
谁登陆了系统
root
```

shell 的数学运算和双小括号

运算符	说明	举例
+	加法	`expr a+a+b` 结果为 30。
-	减法	`expr a-a-b` 结果为 -10。
*	乘法	`expr a\*a\*b` 结果为 200。
/	除法	`expr b/b/a` 结果为 2。
%	取余	`expr bba` 结果为 0。
=	赋值	a=$b 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[a==a==b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[a!=a!=b] 返回 true。

```
[root@localhost shell_learning]#
[root@localhost shell_learning]# echo $((8>7))
1
[root@localhost shell_learning]# echo $((8==7))
0
[root@localhost shell_learning]#
```

关于 shell 逻辑运算 返回 1 或 0 的值

实践双小括号用法,加减乘除

```
[root@localhost shell_learning]# echo $((3+4))
7
[root@localhost shell_learning]# echo $((3-4))
-1
[root@localhost shell_learning]# echo $((3*4))
12
[root@localhost shell_learning]# echo $((3/4))
0
[root@localhost shell_learning]# echo $((10/3))
3
[root@localhost shell_learning]# echo $((5/3))
1
[root@localhost shell_learning]# echo $((5**2))
25
[root@localhost shell_learning]# echo $((5**4))
625
[root@localhost shell_learning]# echo $((7%4))
3
[root@localhost shell_learning]#
```

加减乘除和取余

综合变量的计算

```
[root@localhost shell_learning]#
[root@localhost shell_learning]#
[root@localhost shell_learning]# num=5
[root@localhost shell_learning]# ((num*3))
[root@localhost shell_learning]# $ num
bash: $: 未找到命令...
[root@localhost shell_learning]# echo $num
5
[root@localhost shell_learning]# ((num=num*3))
[root@localhost shell_learning]# echo $num
15
[root@localhost shell_learning]# echo $((num=num*3))
45
[root@localhost shell_learning]#
```

特殊符号运算

++ 加一

– –减一

++a 先计算 +1 然后再赋值给变量 a

a++ 先对变量 a 操作 然后再加 1

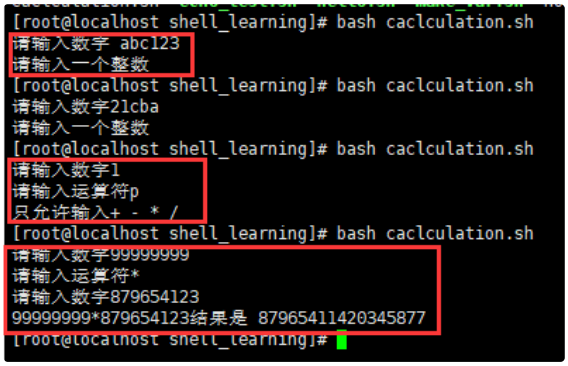
```
[root@localhost ~]# a=6
[root@localhost ~]# echo $a((++a))
-bash: 未预期的符号 `(' 附近有语法错误
[root@localhost ~]# echo $((++a))
7
[root@localhost ~]# echo $a
7
[root@localhost ~]# echo $((a++))
7
[root@localhost ~]# echo $a
8
[root@localhost ~]#
```

shell 脚本实践

函数的作用,就是把你写的功能代码,进行打包,封装成函数,方便调用执行

根据输入的数字进行计算

```bash
#!/bin/bash

#脚本实践

#函数的作用,就是把你写的功能代码,进行打包,封装成函数,方便调用执行
print_usage(){
        printf "请输入一个整数\n"

        exit 1
}
#接受用户输入的命令 -p参数后面写给用户看到的提示信息
#read -p "提示信息" 接受用户输入的变量
read -p "请输入数字"    firstnum

#使用if进行逻辑判断
#限制用户输入纯数字
if [ -n "`echo $firstnum|sed 's/[0-9]//g'`" ]
   then
        print_usage
fi

#再对运算符进行输入
read -p "请输入运算符" operrator

#在对运算符进行判断
#限制在 + - * / 四个符号

if [ "${operrator}" != "+" ] && [ "${operrator}" != "-" ] && [ "${operrator}" != "*" ] && [ "${operrator}" != "/" ]
   then
        echo"只允许输入+ - * /"
        exit 2
fi

#对第二个输入的数字 进行判断
read -p "请输入数字" secondnum
if [ -n "`echo $secondnum|sed 's/[0-9]//g'`" ]
   then
        print_usage
fi

#数值计算
echo "${firstnum}${operrator}${secondnum}结果是 $((${firstnum}${operrator}${secondnum}))"

```



最终运行效果 正常判断数字 判断运算符 过滤字母 给出提示

使用脚本判断 nginx 是否存活

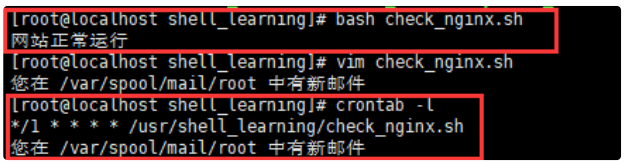
```bash
#!/bin/bash


checkurl(){

   timeout=5
   #相当于定义一个计数器
   fails=0
   success=0


   #循环检测
   while true
        do
           wget --timeout=${timeout} --tries=1 http://192.168.70.17/ -q -O /dev/null
           #echo $?
           # if条件参数 -ne 代表不等于
           if [ $? -ne 0  ]
                then
                   let fails=fails+1 #代表失败次数加1
           else
                   let success+=1 #代表成功次数加1
           fi

           #判断当成功次数大于等于1的时候,可以得出该网站是正常的
           # -ge 是大于等于的意思
           if [ $success -ge 1 ]
                then
                echo "网站正常运行"
                #返回状态码0
                exit 0
           fi


           #当错误次数>2时候 告警 发邮件等
           if [ ${fails} -ge 2 ];then

              echo "网站一定挂了"
              exit 2
           fi
done
}
checkurl

```

然后把这个脚本添加到定时任务里面

crontab –e

* * * * * /usr/shell_learning/check_nginx.sh

代表每分钟进行命令执行



直接运行 提示网站正常

定时任务里面 每分钟定期发送到 root 里面

```
From root@localhost.localdomain  Wed Aug 17 13:57:01 2022
Return-Path: <root@localhost.localdomain>
X-Original-To: root
Delivered-To: root@localhost.localdomain
Received: by localhost.localdomain (Postfix, from userid 0)
        id 7F2AD2073C2D; Wed, 17 Aug 2022 13:57:01 +0800 (CST)
From: "(Cron Daemon)" <root@localhost.localdomain>
To: root@localhost.localdomain
Subject: Cron <root@localhost> /usr/shell_learning/check_nginx.sh
Content-Type: text/plain; charset=UTF-8
Auto-Submitted: auto-generated
Precedence: bulk
X-Cron-Env: <XDG_SESSION_ID=165>
X-Cron-Env: <XDG_RUNTIME_DIR=/run/user/0>
X-Cron-Env: <LANG=zh_CN.UTF-8>
X-Cron-Env: <SHELL=/bin/sh>
X-Cron-Env: <HOME=/root>
X-Cron-Env: <PATH=/usr/bin:/bin>
X-Cron-Env: <LOGNAME=root>
X-Cron-Env: <USER=root>
Message-Id: <20220817055701.7F2AD2073C2D@localhost.localdomain>
Date: Wed, 17 Aug 2022 13:57:01 +0800 (CST)

网站正常运行
```

expr 命令

必须通过空格传递参数 可进行计算,也可以进行逻辑判断

```
[root@localhost ~]# expr 5 \* 3
15
[root@localhost ~]# expr 5\*3
5*3
[root@localhost ~]# expr 5 \* 3
15
[root@localhost ~]# expr length 456789
6
[root@localhost ~]# expr 5 \>6
expr: 语法错误
[root@localhost ~]# expr 5 \> 6
0
[root@localhost ~]# expr 5 \> 4
1
```

正确返回 1 错误返回 0

还可以进行模式匹配,冒号代表统计字符串,.*代表任意的字符串重复 0 次或者多次

```
[root@localhost ~]# expr chen ":" "."
1
[root@localhost ~]# expr chen.jpg ":" ".*"
8
[root@localhost ~]# expr chen.jpg ":" "^."
1
[root@localhost ~]# expr chen.jpg ":" "*."
0
[root@localhost ~]# expr chen.jpg ":" "*"
0
[root@localhost ~]# expr chen.jpg ":" ".*"
8
```

通过 expr 命令判断文件名后缀是否合法

```
[root@localhost shell_learning]# vim file_houzhui.sh
[root@localhost shell_learning]# bash file_houzhui.sh   你好.jpg
是jpg文件
[root@localhost shell_learning]# bash file_houzhui.sh   你好.jp
其他文件
[root@localhost shell_learning]# vim file_houzhui.sh
```

Bash

```
#!/bin/bash
#判断来自第一个参数以后 统计字符并匹配后缀名
if expr "$1" ":" ".*\.jpg" &> /dev/null
        then
        echo "是jpg文件"
else
        echo "其他文件"
fi
```

找出长度不大于 6 的单词

```bash
#!/bin/bash
#通过for循环 先显示变量这些词组
for  str1 in i am student, i like learn linux
        do
#进行逻辑判断 如果变量str1 字符串长度 >6 (ge 在if判断里代表> lt代表<)
        if [ `expr length $str1`  -ge 6  ]

          then
          #输出变量str1
                echo $str1
fi
done
```

```
[root@localhost shell_learning]# bash length_word.sh
student,
[root@localhost shell_learning]# expr length student
7
```

运行结果 直接显示最长的字符串

bc 命令 — 数字计算器

Bash 解释器仅能够进行整数计算，而不支持浮点运算，因此有时要用到 bc 命令进行高精度的数字计算工作

```
[root@localhost shell_learning]# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.

4*4
16
3.1415926*9
28.2743334
```

在 BC 状态下面可以进行 高精度计算

也可以配合管道符 进行多重计算

计算 1–1000 所有数字加起来的总和

[root@localhost shell_learning]# echo {1..1000} | tr ' ' + | bc
500500

```
[root@localhost shell_learning]# echo {1..1000} | tr ' ' + | bc
500500
```

shell 条件测试

通过 test 命令 检查条件是否成立

```
[root@localhost shell_learning]# ls
caclculation2.sh  check_nginx.sh  file_houzhui.sh  length_word.sh  nohup.out       teshu.sh
caclculation.sh   echo_test.sh    hello.sh         make_var.sh     special_var.sh
[root@localhost shell_learning]# test -e hello.sh
[root@localhost shell_learning]# echo $1

[root@localhost shell_learning]# echo $?
0
[root@localhost shell_learning]# test -e special_var.shhhhh
[root@localhost shell_learning]# echo $?
1
```

正确返回 0 错误返回 1 可以判断是否存在这个文件

test 也可以用于文件的权限侦测和两个文件之间进行比较

执行条件表达式并显示返回值。

```Bash
[root@pc root]$ test ! "abc" == 123; echo ?
 0
# 等价形式
[root@pc root] [ ! "abc" == 123 ]; echo ?
 0
 [root@pc root] [[ ! "abc" == 123 ]]; echo $?
 0
```

-f 判断文件是否是普通文件类型是个话返回 TRUE 不是的话 返回 FALSE

```
[root@localhost shell_learning]# test -f hello.sh && echo "是普通文件" || echo "不是普通文件"
是普通文件
[root@localhost shell_learning]# test -f nohup.out && echo "是普通文件" || echo "不是普通文件"
是普通文件
[root@localhost shell_learning]# mkdir hello
[root@localhost shell_learning]# test -f hello && echo "是普通文件" || echo "不是普通文件"
不是普通文件
```

-d 判断是否是目录类型 是个话返回 TRUE 不是的话 返回 FALSE

```
[root@localhost shell_learning]# ls
caclculation2.sh  check_nginx.sh  file_houzhui.sh  hello.sh        make_var.sh  special_var.sh
caclculation.sh   echo_test.sh    hello            length_word.sh  nohup.out
[root@localhost shell_learning]# test -d hello && echo"ok" || echo "no"
bash: echook: 未找到命令...
no
[root@localhost shell_learning]# test -d hello && echo ok || echo no
ok
```

-z 字符串为空,则为 TRUE

-n 字符串不为空 则为 TRUE

```
[root@localhost shell_learning]# test -z "" && echo ok || echo no
ok
[root@localhost shell_learning]# test -z " " && echo ok || echo no
no
[root@localhost shell_learning]# test -n " " && echo ok || echo no
ok
```

中括号条件测试[ ]

```
[root@localhost shell_learning]# file1=hello.sh
[root@localhost shell_learning]# echo $file1
hello.sh
[root@localhost shell_learning]# [ -f "${file1}" ] && echo
caclculation2.sh  check_nginx.sh  file_houzhui.sh  hello.sh        make_var.sh  special_var.sh
caclculation.sh   echo_test.sh    hello/           length_word.sh  nohup.out
[root@localhost shell_learning]# [ -f "${file1}" ] && echo ok || echo no
ok
[root@localhost shell_learning]#
```

通过定义变量 file1 返回测试正确值 ok

双中括号验证文件是否有写入权限

[chen@localhost shell_learning]$ [-r "大碗宽面.txt"] && cat 大碗宽面.txt || echo "你没权限 看个锤子"

```
-rw-r--r--. root root     0 8月  18 14:58 大碗宽面.txt
[chen@localhost shell_learning]$ [ -r "大碗宽面.txt" ] && cat 大碗宽面.txt || echo "你没权限 看个锤子"
[chen@localhost shell_learning]$ [ -r "大碗宽面.txt" ] && cat 大碗宽面.txt || echo " 看个锤子"
[chen@localhost shell_learning]$ [ -w "大碗宽面.txt" ] && cat 大碗宽面.txt || echo " 你没权限,你写个锤子"
 你没权限,你写个锤子
[chen@localhost shell_learning]$ [ -w "大碗宽面.txt" ] && (echo "你看这个 它又大又圆">大碗宽面.txt) || echo " 你没权限,
你写个锤子"
 你没权限,你写个锤子
[chen@localhost shell_learning]$ chmod +w 大碗宽面.txt
chmod: 更改"大碗宽面.txt" 的权限: 不允许的操作
[chen@localhost shell_learning]$ [ -w "大碗宽面.txt" ] && (echo "你看这个 它又大又圆">大碗宽面.txt) || echo " 你没权限,
你写个锤子"
 你没权限,你写个锤子
```

Bash

```
[root@localhost shell_learning]# chmod a+w 大碗宽面.txt
-rw-rw-rw- 1 root root     0 8月  18 15:44 大碗宽面.txt
```

通过 root chmod 更改其他用户组 写入权限以后 可以正常写入并输出

```
[chen@localhost shell_learning]$ [ -w "大碗宽面.txt" ] && (echo "你看这个 它又大又圆">大碗宽面.txt) || echo " 你没权限
你写个锤子"
[chen@localhost shell_learning]$
[chen@localhost shell_learning]$
[chen@localhost shell_learning]$ cat 大碗宽面.txt
你看这个 它又大又圆
```

字符串的逻辑判断

!= 不等于

= 等于

! 取反

```
[root@localhost shell_learning]# [ "chen" = "chen" ] && echo ok || echo no
ok
[root@localhost shell_learning]# [ "chen" = "cen" ] && echo ok || echo no
no
[root@localhost shell_learning]# name1=chen
[root@localhost shell_learning]# [ "${name1}" = "chen"  ] && echo ok || echo no
ok
[root@localhost shell_learning]# name1=666
[root@localhost shell_learning]# [ "${name1}" = "chen"  ] && echo ok || echo no
no
[root@localhost shell_learning]# [ "${name1}"! = "chen"  ] && echo ok || echo no
no
[root@localhost shell_learning]# [ "${name1}"! = "666"  ] && echo ok || echo no
no
[root@localhost shell_learning]# [ "${name1}"!= "666"  ] && echo ok || echo no
-bash: [: 666!=: 期待一元表达式
no
[root@localhost shell_learning]# [ "${name1}"! = "666"  ] && echo ok || echo no
no
```

数值比较测试

在 [] 以及 test 中使用的比较符号	在 (()) 和 [[]] 中使用的比较符号	说明
-eq	== 或 =	相等，全拼为 equal
-ne	!=	不相等，全拼为 not equal
-gt	>	大于，全拼为 greater than
-ge	>=	大于等于，全拼为 greater equal
-lt	<	小于，全拼为 less than
-le	<=	小于等于，全拼为 less equal

在中括号中,使用数学比较符号,请添加转义符号

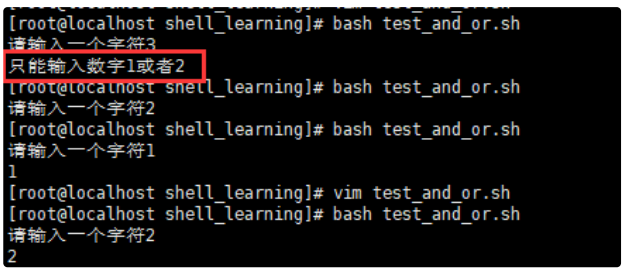
\> 再不添加转义符的情况下 会出现执行错误混乱的一个情况

逻辑运算的脚本学习

接受用户输入,判断它是否等于某个数字

```bash
#!/bin/bash

read -p "请输入一个字符" var1

#逻辑条件测试
 [  "$var1" -eq "1" ] && {
  echo $var1
  exit 0
}

#如果变量 =2 然后输出变量
 [  "$var1" = "2"  ] && {
  echo $var1
  exit  0
}

#除了1,2以外的数字 其他报错

[ "$var1" =≠ "2" -a "$var1" =≠ "1"  ] && {
   echo "只能输入数字1或者2"
  exit 2
}
~
```

执行效果正常 输入 1 和 2 以外的数字 直接过滤出现提示



模拟安装逻辑脚本实践

根据要求输入指令 进行相对应的指令逻辑判断

Bash

```bash
#!/bin/bash

path=/usr/shell_learning/test_scripts

#判断路径 如果没有 按path变量 进行创建
[ ! -d "$path" ]&& mkdir $path -p
#打印end之间的内容
cat << end
   1. [install lamp]
   2. [install lnmp]
   3. [exit]
   "请选择对应数字执行"
end

read num

expr $num+1 &> /dev/null

[ $? -ne 0 ] && {

 echo "请输入1,2,3之内的选项"
 exit 1
}

#对输入的数字1,2,3进行判断
[ "$num" -eq "1" ] && {

 echo "正在安装lamp中"
 sleep 2;

 # 对文件权限进行判断
 [ -x "$path/lamp.sh" ] || {
 echo "没有该文件的执行权限,请联系管理员"
 exit 2
}
 $path/lamp.sh
 exit $?
}


# 判断选择2的情况,安装lnmp

[ "$num" -eq "2" ] && {
  echo "正在安装lnmp"
  sleep 2;

# 对文件执行权限判断

 [ -x "$path/lnmp.sh"  ] || {
 echo "没有该文件的执行权限,请联系管理员"
 exit 2
}
 $path/lamp.sh
 exit $?
}
# 退出

[ "$num" -eq 3 ] && {
  echo "退出"
  exit 3
}
#判断数字编号
 [[ ! "$sum" =~ [1-3] ]] && {
  echo "请输入指定数字"
  exit 4
}

```

```
68
```

```
[root@localhost test_scripts]# bash lamp_or_lnmp.sh
   1. [install lamp]
   2. [install lnmp]
   3. [exit]
   "请选择对应数字执行"
5
请输入指定数字
[root@localhost test_scripts]# bash lamp_or_lnmp.sh
   1. [install lamp]
   2. [install lnmp]
   3. [exit]
   "请选择对应数字执行"
1
正在安装lamp中
lamp is installed
[root@localhost test_scripts]# bash lamp_or_lnmp.sh
   1. [install lamp]
   2. [install lnmp]
   3. [exit]
   "请选择对应数字执行"
3
退出
[root@localhost test_scripts]#
```

执行效果 能正常根据数据要求进行逻辑判断 安装以及退出

if 语句学习,

if <条件表达式>

then

代码....

if

then

代码....

fi

fi

if <条件表达式>

then

当条件成立 ,进行执行

else

否则执行这个

fi

if 实践

可以用 if 语句代替部分正则表达式

Bash

```bash
#!/bin/bash

if  [ -f /etc/hosts ]
        then
        echo "[  ] it is ok"

fi

if  [[ -f /etc/hosts ]] ; then
        echo "[[]] it is ok"

fi

if  test -f /etc/hosts ; then
        echo "test it is ok"
fi
```

通过 if 判断参数大小

```bash
#!/bin/bash

a=$1
b=$2

if [ "$a" -lt "$b" ];then
        echo "yes, $a 小于 $b "
        exit 0
fi

if [ "$a" -eq "$b" ];then
        echo "yes, $a 等于 $b "
        exit 0
fi

if [ "$a" -gt "$b" ];then
        echo "yes, $a 大于 $b"
        exit 0
fi
~
输出的结果是
[root@localhost test_scripts]# bash read_if.sh 5 4
yes, 5 大于 4
[root@localhost test_scripts]# bash read_if.sh 5 5
yes, 5 等于 5
[root@localhost test_scripts]# bash read_if.sh 4 5
yes, 4 小于 5

```

上面是单分支的 IF 脚本语句 可以通过多分支进行简化

```bash
#!/bin/bash

a=$1
b=$2

if [ "$1" -le "$2"  ]; then
        echo "是的,$1 小于 $2"

elif [ "$1" -eq "$2"  ]; then
        echo "是的,$1 等于 $2"

elif [ "$1" -gt "$2" ]; then
        echo "是的,$1 大于 $2"

fi
~
输出的结果
[root@localhost test_scripts]# bash read_if2.sh 5 6
是的,5 小于 6
[root@localhost test_scripts]# bash read_if2.sh 4 3
是的,4 大于 3
[root@localhost test_scripts]# bash read_if2.sh 3 3
是的,3 小于 3

```

linux 检测内存实践

检测 linux 剩余内存,当内存小于 100m 发邮件给运维

并且将脚本设置为 3 分钟执行一次,自动检测内存

思路:

1.获取内存情况

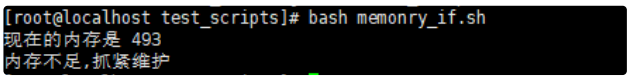
2.配置邮件告警,发送告警邮件

3.判断剩余内存,是否需要发送邮件

4.加入 crontab 3 分钟执行一次

Bash

```
#!/bin/bash
#定义一个变量free_memory 获取 freem-m 内存的最后一行
free_memory=`free -m |awk 'NR==2 {print $NF}'`
#定义chars变量 并输出free_memory
chars="现在的内存是 $free_memory"
#进行判断 如果当前变量内存 小于 2100
if [ "$free_memory" -lt "2100"  ]
        then      #输出当前内存到messages.txt中
                echo $chars|tee /tmp/messages.txt
        #邮件 主题,收件人,内容
        mail -s "`date +%F-%T`$chars" 200921743@qq.com < /tmp/messages.txt
        echo "内存不足,抓紧维护"

fi

[root@localhost test_scripts]# bash memonry_if.sh
现在的内存是 493
内存不足,抓紧维护

并且加入计时任务
[root@localhost test_scripts]# crontab  -l
*/3 * * * * /usr/shell_learning/test_scripts/memonry_if.sh  &>/usr/shell_learning/memory_test.txt


每隔3分钟执行一次,并且将日志 输出到memory_test.txt里面

编辑自动发送邮件的文档

vi /etc/mail.rc

添加如下信息

# For Linux and BSD, this should be set.

set bsdcompat

set from=200921743@qq.com smtp=smtp.qq.com

set smtp-auth-user=alvin smtp-auth-password=123456 smtp-auth=login

#注意：200921743@qq.com是邮箱账号，alvin 是我的邮箱用户名，123456 是邮箱密码

```



学习 Python 和 php 连接 mysql

先通过 yum install php-mysqlnd php 安装相关环境依赖

在安装好依赖环境以后 开始制作 conn 进行连接

```Bash
<?php
$mysql_id=mysql_connect("localhost","root","123456") or mysql_error();
if ($mysql_id){
    echo "mysql连接成功\n";
} else{
 echo mysql_error();
}
当运行脚本的时候 发生如下错误
[root@localhost test_scripts]# php mysql_php.php
PHP Warning:  mysql_connect(): The server requested authentication method unknown to the client [caching_sha2_password] in /usr/shell_learning/test_scripts/mysql_php.php on line 2
PHP Warning:  mysql_connect(): The server requested authentication method umknown to the client in /usr/shell_learning/test_scripts/mysql_php.php on line 2

后面查证mysql8默认的使用密码认证方式不一样，mysql8.0默认使用caching_sha2_password，但是之前版本都是使用mysql_native_password

解决方案：
将密码认证方式caching_sha2_password修改为mysql_native_password
mysql> select host,user,plugin from user;
+-----------+------------------+-----------------------+
| host      | user             | plugin                |
+-----------+------------------+-----------------------+
| localhost | mysql.infoschema | caching_sha2_password |
| localhost | mysql.session    | caching_sha2_password |
| localhost | mysql.sys        | caching_sha2_password |
| localhost | root             | mysql_native_password |
+-----------+------------------+-----------------------+

[root@localhost test_scripts]# php mysql_php.php
mysql连接成功
重新运行脚本 提示连接成功


```

python 连接数据库

安装对应的 Python3 环境依赖

yum install python3 python3-devel python3-pip

安装链接 sql 模块

pip3 install pymysql

```Bash
import pymysql
#定义变量conn
conn = pymysql.connect(
#主机名
host='localhost',
#端口
port=3306,
#用户
user='root',
#密码
password='123456',
db='mysql',
charset='utf8'
)
#操纵数据库
cursor=conn.cursor()
cursor.execute('select version()')

data=cursor.fetchone()

print("数据库连接成功,当前版本数据库是:%s"%data)
conn.close()
在运行的时候出现如下问题

CryptographyDeprecationWarning: Python 3.6 is no longer supported by the Python core team. Therefore, support for it is deprecated in cryptography and will be removed in a future release.
虽然不影响正常使用 但是每次都会出现这个提示
经查证发现是因为高版本的cryptography的原因 可能会导致mysql8.0依赖的api会有问题
通过pip3 list 发现是安装的是最新版的cryptography 37.0.2
移除37.0.2版本 安装36.0.2版本
正常运行无提示
[root@localhost test_scripts]# python3 mysql_python.py
数据库连接成功,当前版本数据库是:8.0.30


```

通过 shell 脚本检测 mysql 是否有在运行

```bash
#!/bin/bash
模拟web服务器 通过mysql账户进行连接
mysql -u root -p 123456 -e "select version();" &>/dev/null
#如果不等于1 启动服务
if [ $? -ne 1 ];then
    #启动服务
    systemctl start  mysqld.service
else
    echo "mysql 正在运行!"
fi
运行效果
[root@localhost test_scripts]# bash check_2.sh
Enter password:
mysql 正在运行!


方法2
#!/bin/bash
#通过判断 进程的状态
ps -ef | grep mysqld| grep -v grep &> /dev/null
#如果返回值是等于0
if [ $? -eq 0 ]
then
        echo "mysql服务正常运行！！！"
else
        echo "mysql服务已经停止！请及时解决！！！"
fi

代码运行情况
[root@localhost test_scripts]# bash check_mysql.sh
mysql服务正常运行！！！
可以根据需求进行 每小时的 或者每分钟的任务执行 如果服务停止则可以进行邮件报警
```

rsync 起停脚本学习

rsync 是 linux 系统下的数据镜像备份工具。使用快速增量备份工具 Remote Sync 可以远程同步，支持本地复制，或者与其他 SSH、rsync 主机同步。
已支持跨平台，可以在 Windows 与 Linux 间进行数据同步。

正常启动启动 rsync 脚本非常麻烦

[root@localhost test_scripts]# /usr/bin/rsync --daemon --config=/etc/rsyncd.conf

而且 rsyncd.conf 文件需要根据系统路径进行更换

无法通过其他服务 例如 mysqld network start stop restart 进行切换

Usage: /etc/init.d/network {start|stop|status|restart|force-reload}

```bash
#!/bin/bash
#如果当前获取的参数个数 等于1
#输出使用方法 start stop restart
if [ "$#" -ne 1  ]; then
   echo "使用方法:$0{ start | stop | restart }"
   exit 1

fi

#当用户选择启动rsync

if [ "$1" = "start" ]; then
   /usr/bin/rsync --daemon
   sleep 2

# 验证端口是否已经启动 如果启动 输出服务已启动
if [ `netstat -tunlp | grep rsync | wc -l` -ge 1 ]; then
  echo "rsync服务已启动"
  exit 0
fi

# 判断第二个情况 如果 参数=stop 则杀死reync 等进程组 把记录消息放进黑洞中
elif [ "$1" = "stop" ]; then
   killall rsync &> /dev/null
   sleep 2
# 验证端口是否停止 如果等于0 则显示端口关闭
if [ `netstat -tunlp | grep rsync | wc -l` -eq 0 ]; then
   echo "rsync服务已关闭"
  exit 0
fi
# 判断第三个情况 如果 参数=restart 则先杀死进程组 rsync 然后重新启动 赋予变量 然后进行判断 输出文字
elif [ "$1" = "restart" ]; then
  killall rsync
  sleep 1
  /usr/bin/rsync --daemon
  kill=`netstat -tunlp | grep rsync | wc -l`

  start=`netstat -tunlp | grep rsync | wc -l`
  if [ "$kill -eq 0" -a "$start" -ge 1 ];then

echo "rsync 服务正在重启"
  exit 0
fi
else
  echo "使用方法:$0{ start | stop | restart }"
fi

遇到的问题
之前在杀进程的时候 使用了pkill命令 发现杀完进程以后 没有执行echo的输出 后面将pkill 改成killall以后正常
以及表达式中 多空格 少空格问题


```

代码运行结果

shell 函数学习

函数的特点,能简化 linux 命令,让整个命令更容易读.容易用

将命令组合起来 就成了函数体

然后还需要给函数体起名字 就是函数名

正常使用函数的话 就使用这个函数名 即可

```Bash
sayhello(){
  echo "你好 你好"
  echo "你好 你好"
  echo "你好 你好"
  echo "你好 你好"
  echo "你好 你好"
 }
 #直接调动函数 并执行
 sayhello
 增加开发效率
```

给脚本传入参数,检测 URL 是否正常

普通 shell 脚本方法

```bash
#!/bin/bash

if [ "$#" -ne 1 ];then
  echo "请输入正确的网址检测"
   exit 1
fi


#利用 wegt 检测url存活性,最大尝试连接次数表1 设定响应超时时间是5 $1代表url参数
wget --spider -q -o /dev/null --tries=1 -T 5 $1

#判断参数值是0 显示 网址正常
 if [ $? -eq 0  ];then
   echo "$1,网站正在运行"
   exit 0

else
#否则 返回参数值1 显示 你网站挂了
   echo "$1,你网站挂了"
   exit 1
fi
执行效果
[root@localhost test_scripts]# bash check_url.sh www.baidu.com
www.baidu.com,网站正在运行
[root@localhost test_scripts]# bash check_url.sh www.baidu.com1
www.baidu.com1,你网站挂了
[root@localhost test_scripts]# bash check_url.sh
请输入正确的网址检测

```

用函数的方式

```bash
#!/bin/bash
#定义函数 usage
function usage(){


if [ "$#" -ne 1 ];then
  echo "请输入正确的网址检测"
   exit 1
fi
}
#定义函数 check_url
check_url(){

#利用 wegt 检测url存活性,最大尝试连接次数表1 设定响应超时时间是5 $1代表url参数
wget --spider -q -o /dev/null --tries=1 -T 5 $1

#判断参数值是0 显示 网址正常
 if [ "$?" -eq 0  ];then
   echo "$1,网站正在运行"
   exit 0

else
#否则 返回参数值1 显示 你网站挂了
   echo "$1,你网站挂了"
   exit 1
fi
}
#将函数接入方法入口 判断如果获取的参数=1 则执行函数usage
main(){
if [ "$#" -ne 1 ]; then
  usage
fi
  check_url $1
}
#调用执行main方法
main $*

执行效果
[root@localhost test_scripts]# bash check_url.sh www.baidu.com
www.baidu.com,网站正在运行
[root@localhost test_scripts]# bash check_url.sh www.baidu.com1
www.baidu.com1,你网站挂了
[root@localhost test_scripts]# bash check_url.sh
请输入正确的网址检测
```

用函数制作 rsync 脚本

Bash

```bash
#!/bin/bash
#定义使用方法
function usage(){
    echo "使用方法:请输入start|stop|restart,来启动服务"
    exit 0
}

#定义start功能
function start(){
  /usr/bin/rsync --daemon
  sleep 2
if [ `netstat -tunlp | grep rsync |  wc -l` -ge 1 ]; then
   echo " rsync已启动 "
else
   echo "rsync没有启动"
fi
}
#定义stop功能
function stop(){
  killall rsync
  sleep 2
if [ `netstat -tunlp | grep rsync | wc -l` -eq 0 ]; then
   echo "rsync已停止"
else
   echo "rsync没有停止"
fi
}
#定义重启服务
function restart(){
  echo ""
}
#定义函数main 如果start 调用start 如果stop调用stop 如果是restart 先调用stop 再选择start
function main() {
if [ "$#" -ne 1 ]; then
  usage
fi

if [ "$1" = "start" ]; then
  start

elif [ "$1" = "stop" ]; then
  stop

elif [ "$1" = "restart" ]; then
  stop
  sleep 1
  start
else
  usage
fi
}
#调动程序入口函数
main $*

代码运行结果
[root@localhost init.d]# /etc/init.d/funct_rsync.sh start
 rsync已启动
[root@localhost init.d]# /etc/init.d/funct_rsync.sh stop
rsync已停止
[root@localhost init.d]# /etc/init.d/funct_rsync.sh restart
rsync: no process found
rsync已停止
 rsync已启动


遇到的问题

```

```
68  中括号表达式中[参数空格问题]
    以及写restart函数的时候 又跟普通shell脚本一样 先写杀进程 然后 再写重启 再进行判断 其实可以直接调用stop和start 来实现重启
```