1. 查看linux系统支持的shel

cat /etc/shells

1. 查看linux默认的shell

echo $SHELL

grep root /etc/passwd

1. 查看系统是否存在漏洞

env x='() { :;};echo be careful' bash -c "echo this is a test"

修复漏洞 升级bash

yum -y update bash

检查

rpm -qa bash

1. 查看当前系统设置的变量

env 只显示全局变量

set 输出所有变量 包括全局变量和局部变量 -o 输出bash shell所有参数配置信息

declare 输出所有的变量、函数、整数和已经导出(export) 的变量

1. 执行脚本的方式和区别

⭐ bash script-name.sh 和 sh script-name.sh 指定bash解释器执行脚本，脚本不需要要执行权限

⭐ ./script\_name.sh 或者path/script\_name.sh 执行，脚本需要具备执行权限

⭐source 或者 . 加载脚本，会读取脚本中的变量到当前用户得环境下，使其在当前shell和父shell 中生效。

1. 设置环境变量的三种方法

export 变量名=value

变量名=value;export 变量名

declare -x 变量名=value

1. 取消变量

unset 变量名

1. 让环境变量永久生效的配置文件

用户的环境变量配置

/userHome/.bashrc ---推荐在此文件中优先设置

/userHome/.bash\_profile

全局环境变量的配置

/etc/profile

/etc/bashrc ---推荐在此文件中优先设置

/etc/profile.d/ --若要在登陆后初始化或者显示加载内容 把脚本放在该目录下 无需加执行权限

1. 设置登陆提示的两种方式

⭐ 在/etc/motd 文件里添加 要提示的内容内容

⭐ 在/etc/profile.d增加脚本，脚本内容是要提示的内容

1. 环境变量初始化与对应脚本得加载顺序

用户登陆：

⭐ 加载/etc/profile全局变量文件

⭐加载/etc/profile.d/目录下得脚本文件

⭐加载/$HOME/.bash\_profile 用户环境变量文件

⭐找到/$HOME/.bashrc 用户环境变量文件 如果有则加载 没有则不执行

⭐继续找/etc/bashrc 全局变量文件 如果有则加载 没有 则不执行

非登陆时启动

⭐只会加载$HOME/.bashrc 用户环境变量文件

⭐找/etc/bashrc 全局环境变量文件 如果有则加载 没有不执行

因此:

最后将用户得环境变量保存在$HOME/.bashrc 文件下，这样在任何情况下都可以执行。

11、shell中得特殊位置参数变量

⭐ $0 获取当前执行脚本得脚本名字 （如果脚本中包含路径 ，则$0中包含路径）

如果想单独获取路径 dirname

如果单独获取脚本名 basename

⭐ $n 表示脚本后第n各参数，如果n>10 最好表示为 ${10}

⭐ $# 当前脚本后跟得参数总个数

⭐ $\* 获取脚本后所有参数拼接值 “$#” 相当于”$1 $2 $n”

⭐ $@ 获取脚本后所有参数 “$@” 相当于 “$1” “$2” “$n”

1. Shell中得特殊状态变量

⭐$? 获取执行上一个指令得执行状态返回值 通常是0 和 1 一般情况下是 exit 返回的值，函数是return返回的数值 最最常用

⭐ $$ 或者当前执行shell脚本的线程号 不常用

⭐ $! 获取上一个在后台工作的进程号 不常用

⭐ $\_ 获取当前脚本的最后一个参数 不常用

1. shell子串处理介绍

字符串 parameter=”asdefgfaasfdewa”

${#paramter} 获取字符串的长度

${parameter:offset} 获取从第offset位置的字符开始截取到末尾的字符

${parameter:offset:length} 获取从第offset位置数 的 length长度个字符

${parameter#word}从头开始删除匹配word的最短字符串

${parameter##word}从头开始删除匹配word的最长字符串

${parameter%word}从尾开始 删除匹配word最短字符串

${parameter%%word}从尾开始 删除匹配word最长字符串

${parameter/pattern/string}使用string替换第一个匹配的pattern

${parameter//patern/string}使用string替换所有匹配的pattern

1. shell扩展变量

${parameter:-word} 如果parameter为空或者未赋值 返回word字符串代替变量值

${parameter:=word}如果paramter为空或者未赋值 将这个变量设置未word值，并返回变量值

${parameter:?word}如果paramter为空或者未赋值 则将word作为标准错误输出

${parameter:+word}如果paramter为空或者未赋值 什么也不做，否则用word值代替变量值。

1. 数值运算 (())

((i=i+2)) 用作赋值运算

i=$(i+2) 用作赋值运算

((7>6&&5==5)) 用作逻辑运算

echo $((2+1)) 打印计算后的结果

注意:只适用于整数的加、减、乘、除运算。

1. let 赋值表达式

let i=i+9 //等同于 ((i=i+9))

let success+=1

比较适用于赋值运算 仅适用于整数

1. expr 数值计算

用法：expr var1 [+ - \* /] var2

参数和操作符之间有空格 可用于整数和小数的计算

expr 判断a 是否未整数

expr $a+1

如果$?为0 ,a 为整数 否则 不是。

expr 0+0 结果为1

1. bc 计算数值

echo “1+2”|bc

更倾向于从命令行里面读取要进行计算的数据

1. 条件测试常用语句

|  |  |  |
| --- | --- | --- |
| test <条件表达式> | test后有空格 | test -f file && echo 1 |
| [ <条件表达式> ] | 中括号有空格 | [ -f file ] && echo 1 |
| [[ <条件表达式> ]] | 内中括号有空格 可通配符 | [[ -f file ]] && echo 1 |
| ((<条件表达式>)) | 无空格 常用计算 | ((2>1&&5=5))&&echo 1 |

1. 文件测试表达式

-d 目录存在 [ -d /etc ] && echo 1

-f 文件存在,且为普通文件 [ -f /etc/host ] && echo 1

-e 文件或者目录存在 [ -e /etc/hosts ] && echo 1

-r 文件存在且可读 [ -r /etc/hosts ] && echo 1

-s 文件存在且文件大于0 [ -s /etc/hosts ] && echo 1

-w 文件存在且可写 [ -w /etc/host ] && echo 1

-x 文件存在且可执行 [ -x /etc/inid.d/iptables ] && echo 1

-L 文件存在且为链接文件 [ -L /etc/host ] || echo 0

F1 nt F2 文件F1 比文件F2更新

F1 ot F2 文件F1比文件F2 更旧

1. 字符串测试表达式

|  |  |
| --- | --- |
| -n “字符串” | 字符串不为空 则为真 |
| -z “字符串” | 字符串为空 则为真 |
| “字符串1” = “字符串2” | 判断两个字符串相等 可以使用 == 代替 = 两边必须有空格 |
| {“字符串1” != “字符串2” | 判断两个字符串不相等 可以使用 != 代替 = 两边必须有空格 |

注意：两个字符串变量比较 最好加 “” 比如 “$A” = “$B” 否则结果可能不准确。

1. 整数二元比较操作符

|  |  |  |
| --- | --- | --- |
| 在[] test中使用 | 在(()) 和 [[]] 中使用 | 说明 |
| -eq | == | 相等 |
| -ne | != | 不相等 |
| -gt | > | 大于 |
| -ge | >= | 大于等于 |
| -lt | < | 小于 |
| -le | <= | 小于等于 |

注意：在[]中使用 < > 符号时候 最好转义 \

1. 逻辑操作符

|  |  |  |
| --- | --- | --- |
| [] test 中使用 | 在[[]] (())中使用 | 说明 |
| -a | && | and 两端都为真 |
| -o | || | or 或 一个为真就为真 |
| ! | ! | not 非 取反 |

[ -f /etc/hosts -a -d /etc/init.d ] && echo 1

使用多个[] 判断

[ $m -eq 20 ]||[ $n -lt 90 ] && echo 1

1. if条件语句 单分支

⭐ 第一种写法

if <条件表达式>

then

cmd

fi

⭐ 第二种写法

if <条件表达式>; then

指令

fi

两种写法效果一样 第二种写法的分号相当于换行

1. if条件语句 双分支

if <条件表达式>

then

cmd1

else

cmd2

fi

1. if条件语句 多分支

if <条件表达式>

then

cmd1

elif

then

cmd2

else

cmd3

fi

1. 函数声明

⭐ 标准写法

function 函数名 ()

{

指令…

return n

}

⭐ 简写1 不推荐

function 函数名

{

指令…

return n

}

函数后面不加括号 感觉不好 所以不推荐

⭐ 简写2 推荐

函数名()

{

指令…

return n

}

1. 函数的执行

⭐ 不带参数

函数名

⭐ 带参数

函数名 参数1 参数2

注意：shell的位置参数 $1 $2…$# $? $@都可以作为参数使用。

$0 在方法中 仍然是 函数名 不会是方法名

$\* 代表将调用者的所有参数列表传递给函数

1. case条件语句 语法

case “变量” in

值1 )

指令1…

;;

值2)

指令2…

;;

\*)

指令3….

;;

esac

扩展: 作为判定条件的值，当想要多个数值执行同一段逻辑的话可以这样写。

case “变量” in

值1 | 值2 )

指令1…

;;

值3 | 值 4)

指令2…

;;

\*)

指令3….

;;

esac

30、while 循环语句

语法

while <条件表达式>

do

cmd…

done

当条件表达式成立时 执行cmd

1. until

语法

until <条件表达式>

do

cmd…

done

当条件表达式不成立时 执行cmd

1. 让脚本在后台运行

While1.sh 是一个守护线程 循环脚本。

|  |  |
| --- | --- |
| 用法 | 说明 |
| sh While.sh & | 脚本后台执行 |
| Ctrl+c | 停止执行当前脚本或任务 |
| Ctrl+z | 暂停执行当前脚本或任务 |
| bg | 把当前脚本或任务放到后台执行 |
| fg | 把当前脚本或任务放到前台执行，如果有多个任务，可以使用fg +任务编号调出任务，如果fg 2 调出第二个脚本任务 |
| jobs | 查看当前执行的脚本或任务 |
| kill | 关闭执行的任务 kill % 任务编号 |

扩展：由于网络或者一些意外因素导致远程连接的用户断开连接，那么脚本执行可能不完整，严重的会影响到系统的正常运行。所以我们执行脚本的时候 可以使用

nohup scriptsName.sh & 将脚本置于后台执行，并且用户退出后仍继续运行

1. for循环语法结构

⭐ 第一种语法结构

for 变量名 in 变量取值列表

do

指令…..

done

⭐ 第二种语法结构

for ((exp1; expr2; expr3))

do

指令….

done

1. linux获取随机数的6种方式

⭐ 第一种

通过系统变量获取 $RANDOM

如：echo $RANDOM

⭐ 第二种

通过openssl 获取

openssl rand -base64 64

openssl rand -base64 8

获取64（8）位随机字符

⭐ 第三种

通过时间(date)获取随机数

date +%s%N

⭐ 第四种

通过 /dev/urandom 配合 cksum 生成随机数

head /dev/urandom|cksum

⭐ 第五种

通过UUID生成随机数

cat /proc/sys/kernel/random/uuid

⭐ 第6种（适用于随机密码的生成）

使用 expect 附带的 mkpasswd 生成随机数

先安装expect包才可以 “yum install expect -y”

mkpasswd -l 11 -d 5 -c 3 -C 2 -s 1

参数意义

-l # (length of password, default = 11) 指定密码长度

-d # (min # of digits, default = 5) 指定密码中至少数字的数量

-c # (min # of lowercase chars, default = 3) 指定密码中至少小写字母数量

-C # (min # of uppercase chars, default = 2) 指定密码中至少大写字母数量

-s # (min # of special chars, default = 1) 指定密码中至少特殊字符数量

-v (verbose, show passwd interaction)

-p prog (program to set password, default = passwd)

34、select循环语句

用法

select 变量名 in 菜单取值列表

do

指令…..

done

扩展:

PS3=”please choose service numer：” PS3是输入提示符 默认是#? 不友好

$REPLY 代表的选项对应的数字

Note:通常与case一起使用

1. break、countine、exit、return 的区别和对比

|  |  |
| --- | --- |
| 命令 | 说明 |
| break n | 如果省略n 退出整个循环 n 跳出层数 |
| Continue | 省略n 代表 忽略本次循环 n 表示退出第n层继续循环 |
| exit n | 退出当前shell程序 n 也可省略，在下一个shell中 通过$?获取 |
| return n | 用于接收函数的返回值 |

1. shell数组

**数组的定义:**

方法一

array=(value1 value2 value3 …)

方法二 key value方式

array=([key1]=value1 [key2]=value2 [key]=value3)

方法三

array[0]=value1; array[1]=value2;array[3]=value3 ….

方法四 动态创建数组

array=($(命令)) 或者 array=(`命令`)

**数组的打印:**

echo ${array[index]} 根据数组下标读取数组元素

echo ${array[\*]} 读取数组的所有元素

echo ${array[@]} 读取数组的所有元素

echo ${#array[\*]} 打印数组元素的个数

数组的赋值:

array[index]=value 为指定下标的元素赋值

**数组的删除：**

unset array[index] 即可清空指定数组的内容 不带下标会清空整个数组

**数组内容的截取和替换：**

array=(1 2 3 4 5)

echo ${array[@:1:3]} 从下标为1 的元素开始截取 截取三个元素

echo ${array[@]/1/b} 把数组中的1替换成b 原数组不变

字符串的一些操作比如 #、## 、%、%%，/ 等同样适用于数组元素

array=(one two three four five)

echo ${array[\*]#o\*} 从左边最短匹配数组元素的 o\* 的数组元素并删除

echo ${array[\*]##o\*} 从左边最长匹配数组元素的 o\* 的数组元素并删除

echo ${array[\*]%f\*} 从右边最短匹配数组元素的 f\* 的数组元素并删除

echo ${array[\*]%%f\*} 从右边最长匹配数组元素的 f\* 的数组元素并删除

1. 使用trap控制信号

我们在执行脚本的时候，可能会不希望由于用户执行了ctrl+c 等强制终止命令带来的系统不安全问题，可以通过trap 为指定的信号 发生时 设置一些操作（比如清理未执行完的任务等），保证运行的稳定性。

kill -a 或者 trap -a 查看系统支持的所有信号（64种）

在使用信号的时候需要省略SIG前缀

企业实战中Linux重要的信号及说明

|  |  |
| --- | --- |
| 信号 | 说明 |
| HUP(1) | 挂起 通常因终端掉线或用户退出而引发 |
| INT(2) | 中断 通常因Ctrl+c 组合键引发 |
| QUIT(3) | 退出，通常因Ctrl+\ 组合键引发 |
| ABRT(6) | 中止，通常因某些严重的执行错误引发 |
| ALRM(14) | 报警，通常用来处理超时 |
| TERM(15) | 终止，通常在系统关机时发送 |
| TSTP(20`) | 停止进程的运行 单该信号可以被处理和忽略，通常按Ctrl+z 引发 |

Shell 脚本中既可以使用数字也可以使用名称来代表信号。

trap command signal

conmand 是接受到信号要执行的命令

signal 是接收到的信号

即 trap ‘命令;命令’ 信号名

或 trap ‘命令;命令’ 信号编号

例:

trap ‘echo warning’ INT 当按到Ctrl+c 时候 执行 echo warning 命令

stty -a 可以列出终端信号与键盘的对应信息

trap “” 2 命令为空 表示屏蔽与数字对应的Ctrl+c 命令 。此时遇到Ctrl+c 不会又任何反应

trap “:” 2 回复Ctrl+c 信号

同时处理多个信号：

trap “” 1 2 3 20 15 执行这些数字信号时 什么也不做

trap “:” 1 2 3 20 15 执行这些数字信号 恢复信号功能

trap “” `echo {1..64}` 屏蔽64个信号

1. Except 自动交互功能

spawn 命令

spawn [选项] [需要自动交互的命令或程序]

例如:spawn ssh [root@192.168.164.10](mailto:root@192.168.164.10) uptime

在spawn命令后面，直接加上要执行的命令或者程序（例如这里的ssh命令），除此之外，spawn还支持如下一些选项

-open ：表示启动文件的进程

-ignore：表示忽略某些信号

Expect命令

在Expect 自动交互程序的执行过程中，当使用spawn执行一个命令或程序之后，会提示某些交互式信息，expect命令的作用就是获取spawn命令执行后的信息，看看是否和事先指定的相匹配，一旦匹配上指定内容就执行expect后面的动作。Expect 命令也有一些选项，相对用的较多的是-re,表示使用正则表达式的方式来匹配。

Expect命令的语法:

Expect 表达式 [ 动作 ]

例如：

spawn ssh [root@192.168.33.130](mailto:root@192.168.33.130) uptime

expect “\*password” {send “123456\n”}

expect的高级用法

可以在一个expect中匹配多个不同的字符串，并给出不同的处理动作。

需要借助 exp\_continue 执行。

#!/usr/bin/expect

spawn [root@192.168.164.11](mailto:root@192.168.164.11) uptime

expect {

“yes/no” {exp\_send “yes\r”; expr\_continue}

“\*password” {exp\_send “123456\r” }

}

exp eof

send\_user 用来打印expect脚本的信息，类似shell里面的echo。默认的send,exp\_send命令都是将字符串输出到expect脚本中去。

send\_user “I am a boy!”

send\_user “I love you,lady!”

exit 退出脚本，可以在退出时候做一些清理工作。

如:

exit -onexit {

Send\_user “Good bye!”

}

Expect 程序变量

Set 变量名 变量值

打印变量

Puts $变量名

特殊的位置参数

Expect 中也有类似 shell 中 $0 $1 $2的位置变量

在expect中 $argv 表示参数数组，可以使用 [lindex $argv n] 接收expect脚本参数

N从0开始。 0代表第一个参数

Expect接受参数和shell有些区别。Expect接受参数

Set <变量名> [lindex $argv <param index>]

其中 $argv0 表示脚本的名字 $argc表示脚本参数的个数。

Expect 中的 if语句

If {条件表达式} {

指令

}

If {条件表达式} {

指令

} else

{

指令

}

注意:if 后面要有空格，else 前后都要又空格。

Timeout关键字

Expect脚本中全局的时间变量。脚本执行在设定的时间内，超时 则执行 超时设定的内容，退出脚本。

spawn [root@192.168.164.11](mailto:root@192.168.164.11) uptime

set timeout 30

expect {

“yes/no” {exp\_send “yes\r”; expr\_continue}

“\*password” {exp\_send “123456\r” }

}

Expect timeout {puts “good bye!”}