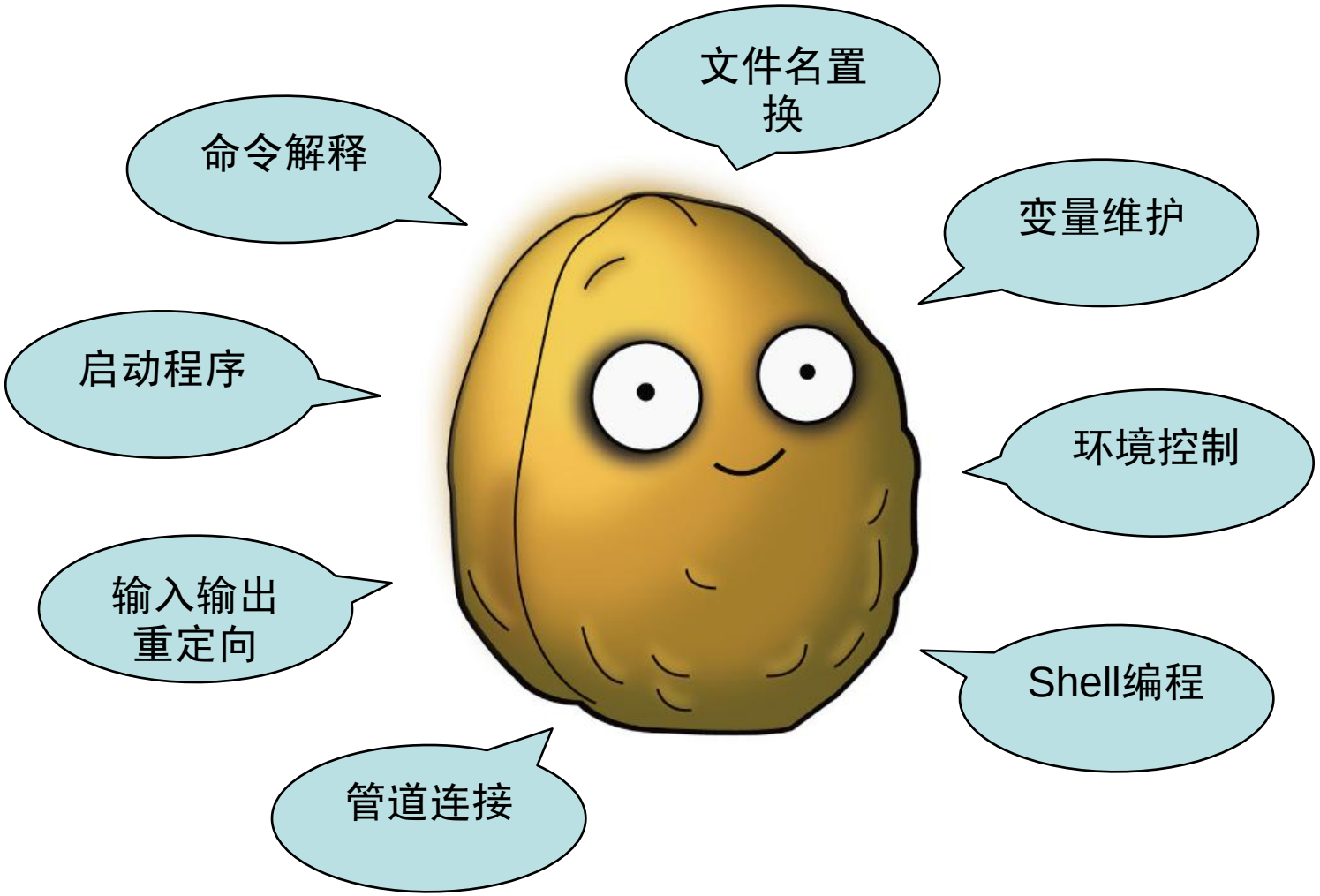
Shell编程技巧和陷阱介绍

INFOP 宋利华
songlihua@baidu.com

WHAT IS SHELL



Challenge

- *.bak -> *
- 对比本机和远程服务器上的配置文件差异

outline

1 - 基础知识

2 - 专项知识

3 - 脚本实践

基础知识

1 - Hello World

2 - 管道

3 - 重定向

4 - 语法结构

Hello World Here Doc

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/bash
echo 'hello,world'
MY_PROGRAM

$ chmod 744 hello.sh
$ ./hello.sh
Hello,world
```

Hello World How to Run

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/bash
echo 'hello,world'
MY_PROGRAM

$ chmod 744 hello.sh
$ ./hello.sh
Hello,world
```

Hello World What to do?

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/bash
echo 'hello,world'
MY_PROGRAM

$ chmod 744 hello.sh
$ ./hello.sh
Hello,world
```

Hello World Executable

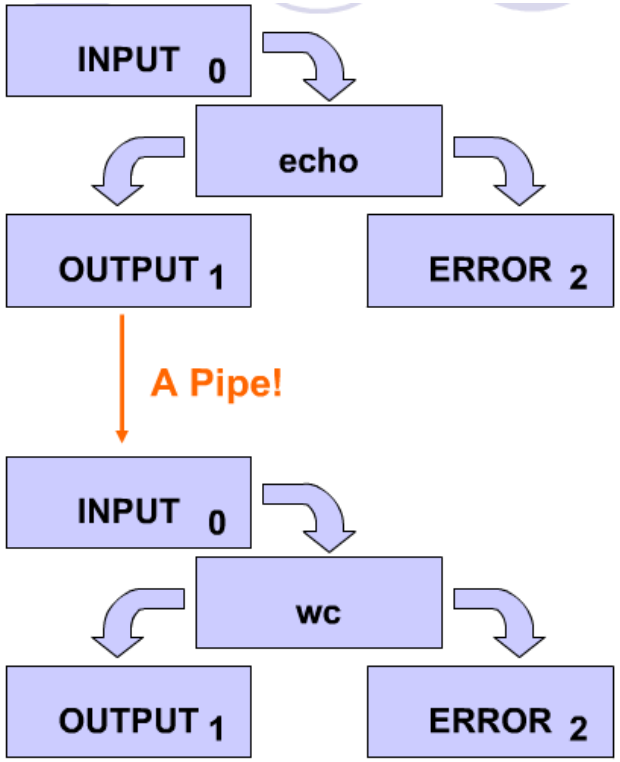
```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/bash
echo 'hello,world'
MY_PROGRAM

$ chmod 744 hello.sh
$ ./hello.sh
Hello,world
```

Hello World Running it

```
$ cat > hello.sh <<MY_PROGRAM
#!/bin/bash
echo 'hello,world'
MY_PROGRAM

$ chmod 744 hello.sh
$ ./hello.sh
Hello,world
```

管道 and 重定向

```
echo hello | wc –c

cat file | tee file.bak | \
grep “something" | \
tee middle_result | \
wc –l

command >/dev/null 2>&1
command &>/dev/null
```



重定向先后顺序

```
$ echo "some text here" > file
$ cat < file
some text here
$ cat < file > file.bak
$ cat < file.bak
some text here
```

问题:

```
$ cat < file > file
$ cat < file
```

输出结果是什么?

输出结果为空

重定向先后顺序

>file 2>&1

和

2>&1 >file

是否有区别？为什么？

分支和循环 – for / if / while

```
for i in `seq 1 3` ; do
    echo $i
done
```

```
while : ; do
    sleep 2
done
```

```
if [ $string == "aaa" ] ; then
    :
elif [ $string == "bbb" ] ; then
    :
else
    :
fi
```

C style for

```
LIMIT=10
for (( a=1 ;
      a<=LIMIT ;
      a++ ))
do
    echo "$a"
done
```

分支和循环 – case / select

```
case $1 in
    1*) echo “1”;;
    2*) echo “2”;;
    *) echo “*”;;
esac
```

```
select var in “a” “b” “c”
do
    break
done
```

调试和知识获取

调试

bash -x 和 echo

知识获取

1.man

2.Abs 《Advanced Bash Scripting Guide》

3.《Unix Power Tools》

outline

1 - 基础知识

2 - 专项知识

3 - 脚本实践

专项知识

1 - 在解释器中

2 - Subshell 父子进程

3 - Substitution 参数、命令、进程替换

4 - 算数与字符串运算

5 - 信号处理

在shell中

- 单引号
- 双引号
- 反引号(推荐使用” $() ”代替” `` ”)
- Awk的引号迷雾

  echo | awk ‘{print “ ’\” ”}’

  echo | awk ‘{print “ \” ”}’
- [ ]和[[ ]]

  在[[ ]]里可以使用&&、||、<、>

  [[ ]]的性能更好

在shell中- test一些常用表达式

-d 目录存在
-e 文件存在
-s 文件长度大于0、非空
-f 正规文件
-w 可写
-L 符号连接
-u 文件有set-user-id位设置
-r 可读
-x 可执行
-z 字符串为空
-n 字符串非空
!判断条件的否定

变量常识

- 变量作用域
  - 全局变量 *（默认全局变量，仅当前shell可见）*
  - 局部变量 *（限定在函数内部，使用local声明）*
- 变量的继承性
  - 使用export导出变量，让子shell可以继承
  - 只能向下单向继承

特殊变量

- 环境变量 (set命名可查看)
  $PATH,$PWD,$LINENO

- 位置参数（Positional Parameters）
  $1 $2 $3 … ${10}

- 特殊参数（Special Parameters）
  $#：位置参数的数量
  $*：所有位置参数的内容 ($1 $2 …)
  $@: 所有位置参数的内容(“$1” “$2” …)
  $?：命令执行后返回的状态
  $$：当前进程的进程号
  $!：后台运行的最后一个进程号
  $0：当前执行的进程名

整数计算

- id++ id-- ++id –id - + ! ~ ** * / % 计算
- << >> & ^ | 位操作
- <= >= < > == != 比较
- && || 逻辑操作
- expr?expr:expr 三元操作符
- = *= /= %= += -= <<= >>= &= ^= |= 赋值操作符

- y=2000 闰年计算
  echo $((y%4==0 && y%100!=0 || y%400==0))
- ((y++)) 自增1
- tmp=$((16#a)) 进制转换

浮点运算

- echo “scale=5; 3/7"|bc
  - .42857
- echo "ibase=16;F"|bc
  - 15
- echo “obase=2;15"|bc
  - 1111
- echo "100.431 20.125" | awk '{printf "%.6f\n",$1*$2}'
  - 2021.173875

Shell中的数组

- 声明数组
  - array[key]=value
  - array=(value1 value2 value3 ...)
  - array=([1]=one [2]=two [3]=three ...)
- 访问数组
  - ${array[key]}
- 数组删除
  - unset array[1]
  - unset array
- 计算数组长度
  - ${#array}

进程替换

- <() >()
  把一个进程的输出回馈给另一个进程
- 同时从文件和标准输入获取：
  command| diff - file2(-代表标准输入)
  另外一种方式
  diff <(command) file2
- 实例：diff两台服务器的同一个配置文件
  vimdiff <(ssh server1 cat conf) <(ssh server2 cat conf)

将所有备份文件.bak还原

dir
 |----as.bak
 |----bs.bak
 |----……
 |----/other_dir/file.bak

备份
```
for file in `ls` ; do
    cp $file $file.bak
done
```

如何还原呢？

内置字符串处理

```
x=aabbaarealwwvvww
echo “${x#a*a}“          # 截去头部最短匹配
bbaarealwwvvww
echo “${x##a*a}“                  # 截去头部最长匹配
lwwvvww
echo “${x%w*w}“                   # 截去尾部最短匹配
aabbaarealwwvv
echo “${x%%w*w}“                  # 截去尾部最长匹配
aabbaareal
```

```
x=abcdabcd
echo ${x/a/b}            # 只替换一个
bbcdabcd
echo ${x//a/b}           # 替换所有
bbcdbbcd
```

```
x=abcdabcd
echo ${x:1}              # 从第x位开始截取至结尾
bcdabcd
echo ${x:1:5}            # 从第x为开始，共截取n位
bcdab
```

将所有备份文件.bak还原

dir
 |----as.bak
 |----bs.bak
 |----……
 |----/other_dir/file.bak

还原

```
for bak_file in `ls *.bak` ; do
    cp $bak_file ${file_bak%%.bak}
done
```

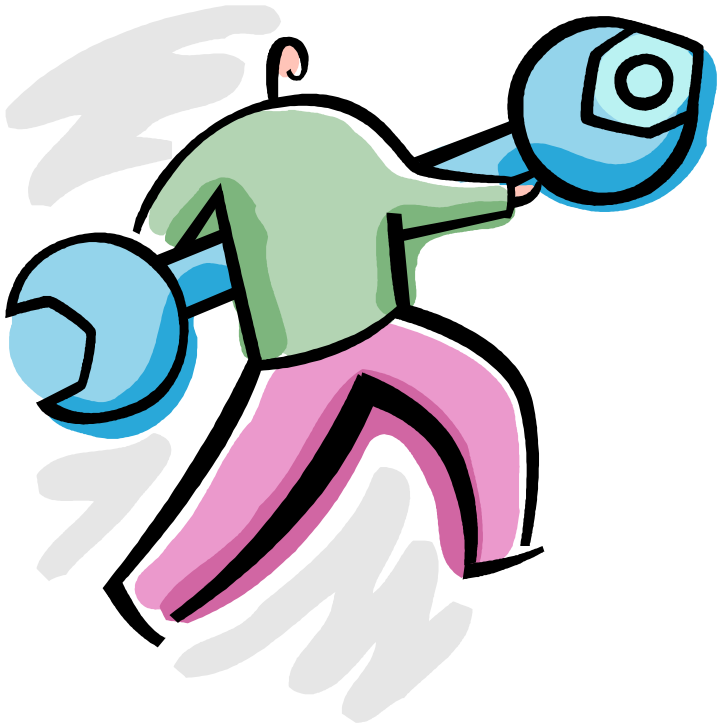
问题

- jx-sys-sdb.jx.baidu.com => jx-sys-sdb
- jx-sys-sdb.jx => tc-sys-sdb.tc

内置字符串处理 – 实际应用

- **截去域名尾部**

```
name=jx-sys-superdb.jx.baidu.com
echo ${name%%.*}
jx-sys-superdb
```



- **替换**

```
echo ${name//jx/tc}
tc-sys-superdb.tc.baidu.com
```

Glob扩展

- 如何扩展：
  - * 任意匹配，包含空串
  - ？匹配单一字符
  - [...] 匹配方括号里的任意1个字符
  - [!...] or [^...] 不匹配方括号里的任意1个字符
  - {str1,str2}匹配包含str1或str2
- 用途：
  - case（多路分支）
  - 内置字符串处理
  - 命令参数

outline

1 - 基础知识

2 - 专项知识

3 - 脚本实践

陷阱一：从ssh返回值说起

- ssh host " true" $?
- ssh host " false" $?
- ssh host " false & " $?

```
ssh host " false & "
if [ $? –eq 0 ]
then
        .......
fi
```

可把返回值存入文件，通过ftp方式统一获取判断。

陷阱一：从ssh返回值说起

- 现象：

  通过ssh执行脚本或者命令后，退出时hang住

- 重现：

  sleep 20 & exit

- 解决方法：

  sleep 20 < /dev/null > /dev/null 2>&1 & exit

- 原因：

  ssh需要确定命令不再有任何输出和输入，并且能有确切的返回值。

陷阱三：ssh的环境变量

- 现象：

  ssh登陆以后java -version版本正确

  ssh username@hostname “java -version”版本不正确
- 解决方法：

  ssh username@hostname “source ~/.bash_profile && java -version"
- 原因：

  ssh登陆时使用的是默认的环境变量.一个比较简单的办法是用source加载一下自己的.bash_profile

陷阱二：脚本遭遇语法错误时

- 现象：

```
if [ $ret -ne 0 ]
then
        echo “NO”
        exit
fi
echo “YES”
```


Thanks