

Shell脚本编程

NSD SHELL

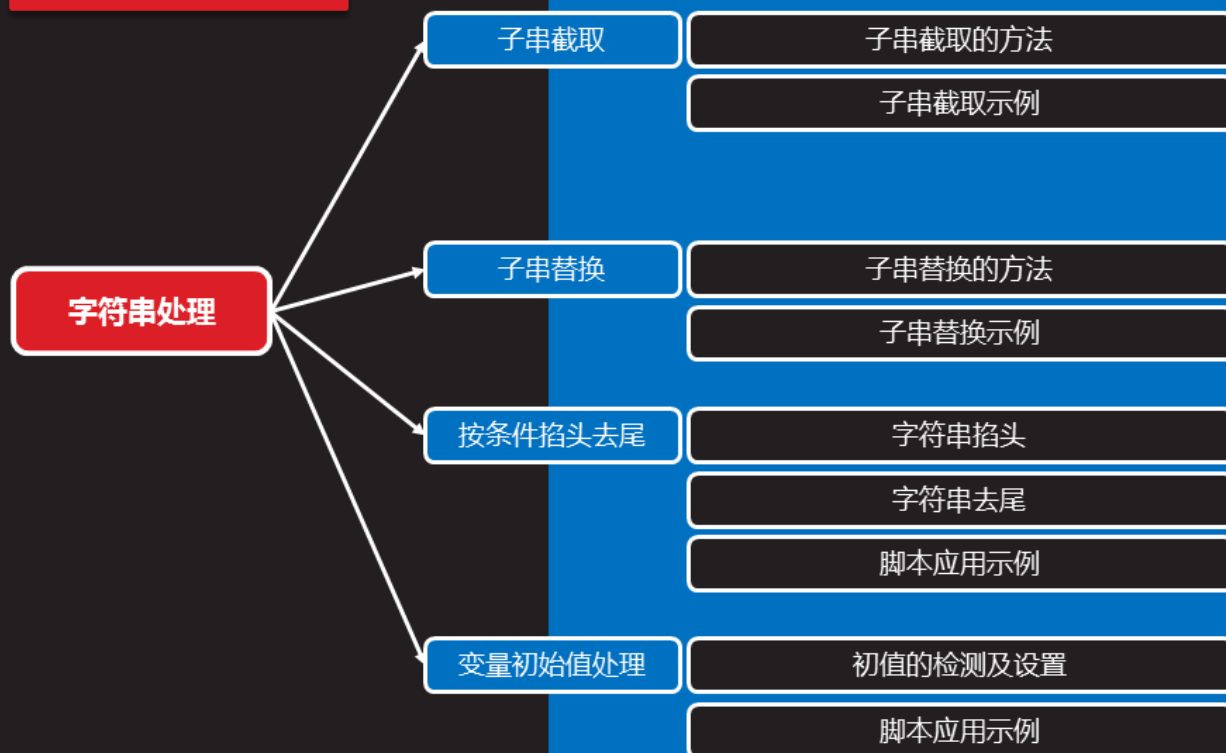
DAY04

内容

上午	09:00 ~ 09:30	作业讲解与回顾
	09:30 ~ 10:20	字符串处理
	10:30 ~ 11:20	
	11:30 ~ 12:00	扩展的脚本技巧
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	正则表达式
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



字符串处理



子串截取

子串截取的方法

- 方法一，使用 `${}` 表达式
 - 格式：`${var:起始位置:长度}`
 - 编号从0开始，可省略
- 方法二，使用 `expr substr`
 - 格式：`expr substr "$var" 起始位置 长度`
 - 编号均从1开始
- 方法三，使用 `cut` 工具
 - 格式：`echo $var | cut -b 起始位置-结束位置`

子串截取示例

知识讲解

- 任务目标

- 截取变量NM的前6个字符

```
[root@svr5 ~]# NM="Tarena IT Group."
```

```
[root@svr5 ~]# echo ${NM:0:6}
```

```
Tarena
```

//采用方法1，与 \${NM::6} 等效

```
[root@svr5 ~]# expr substr "$NM" 1 6
```

```
Tarena
```

//采用方法2

```
[root@svr5 ~]# echo $NM | cut -b 1-6
```

```
Tarena
```

//采用方法3，与 cut -b -6 等效



子串替换

子串替换的方法

知识讲解

- 只替换第1个匹配结果
 - 格式：`${var/old/new}`
- 替换全部匹配结果
 - 格式：`${var//old/new}`



子串替换示例

知识讲解

- 任务目标
 - 将变量NM中的a替换为##

```
[root@svr5 ~]# echo ${NM/a/##}  
T##rena IT Group.
```

//只替换掉第一个 a

```
[root@svr5 ~]# echo ${NM//a/##}  
T##ren## IT Group.
```

//替换掉所有的 a



按条件掐头去尾

字符串掐头

知识讲解

- 从左向右，最短匹配删除
 - 格式：`${变量名}*关键词`
 - 从左向右，最长匹配删除
 - 格式：`${变量名}##*关键词`
- # 用来删除头部，* 通配

```
[root@svr5 ~]# MDIR="/var/spool/mail/root"
[root@svr5 ~]# echo ${MDIR#*/}
var/spool/mail/root                                //删除到最近匹配
[root@svr5 ~]# echo ${MDIR##*/}
root                                                //删除到最远匹配
```



字符串去尾

知识讲解

- 从右向左，最短匹配删除
 - 格式：`${变量名%关键词*}`
 - 从右向左，最长匹配删除
 - 格式：`${变量名%%关键词*}`
- % 用来删除头部，* 通配

```
[root@svr5 ~]# MDIR="/var/spool/mail/root"
[root@svr5 ~]# echo ${MDIR%o*}
/var/spool/mail/ro //删除到最近匹配
[root@svr5 ~]# echo ${MDIR%%o*}
/var/sp //删除到最远匹配
```



脚本应用示例

知识讲解

- 任务目标
 - 实现批量改名，将扩展名 .doc 改为 .txt

```
[root@svr5 ~]# cat renfile.sh
#!/bin/bash
for FILE in *.doc
do
    mv $FILE ${FILE%.doc}.txt
done
```

```
root@svr5:~
[root@svr5 ~]# ls *.doc
file1.doc  xyz.doc
[root@svr5 ~]# ./renfile.sh
[root@svr5 ~]# ls *.txt
file1.txt  xyz.txt
[root@svr5 ~]# ls *.doc
ls: *.doc: 没有那个文件或目录
```



案例1：字符串截取及切割

课堂练习

1. 参考PPT示范操作，完成子串截取、替换等操作
2. 根据课上的批量改名脚本，编写改进版renfilex.sh：
 - 1) 能够批量修改文件的扩展名
 - 2) 修改前/后的扩展名通过位置参数\$1、\$2提供



变量初始值处理

初值的检测及设置

知识讲解

- 取值, `${var:-word}`
 - 若变量var已存在且非Null, 则返回 `$var` 的值
 - 否则返回字符串 "word", 变量var值不变

```
[root@svr5 ~]# NM="Tarena IT Group."
```

```
[root@svr5 ~]# echo ${NM:-Tarena}
```

```
Tarena IT Group.
```

//变量NM已设置

```
[root@svr5 ~]# unset NM
```

//清除NM变量

```
[root@svr5 ~]# echo ${NM:-Tarena}
```

```
Tarena
```

//输出提供的字符串

```
[root@svr5 ~]# echo $NM
```

//前面已清空, 所以无结果



脚本应用示例

知识讲解

- 任务目标
 - 提示输入一个正整数x, 求从1~x的和
 - 若用户未输入值, 则赋初值 `x=1`, 避免执行出错

```
[root@svr5 ~]# cat sumx.sh
```

```
#!/bin/bash
```

```
read -p "请输入一个正整数: " x
```

```
x=${x:-1}; i=1; SUM=0
```

```
while [ $i -le $x ]
```

```
do
```

```
    let SUM+=i ; let i++
```

```
done
```

```
echo "从1到$x的总和是: $SUM"
```

```
root@svr5:~  
[root@svr5 ~]# ./sumx.sh  
请输入一个正整数:  
从1到1的总和是: 1  
[root@svr5 ~]# ./sumx.sh  
请输入一个正整数: 50  
从1到50的总和是: 1275  
[root@svr5 ~]# ./sumx.sh  
请输入一个正整数: 100  
从1到100的总和是: 5050  
[root@svr5 ~]#
```



案例2：字符串初值的处理

编写一个脚本sumx.sh，求从1-x的和：

- 1) 从键盘读入x值
- 2) 当用户未输入任何值时，默认按1计算

课堂练习



扩展的脚本技巧

扩展的脚本技巧

Shell数组

关于变量的类型

定义/赋值数组

输出数组元素

expect预期交互

expect简介

expect应用示例

Shell数组

关于变量的类型

- Shell对变量类型的管理比较松散
 - 变量的值默认均视为文本
 - 用在数学运算中时，自动将其转换为整数

知识讲解

```
[root@svr5 ~]# var1=123
```

```
[root@svr5 ~]# var2=$var1+20
```

```
[root@svr5 ~]# echo $var2
```

```
123+20
```

//123作为文本字符串

```
[root@svr5 ~]# expr $var1 + 20
```

```
143
```

//123作为整数值



定义/赋值数组

知识讲解

- 方法一，整体赋值：
 - 格式：**数组名=(值1 值2 ... 值n)**
 - 示例：SVRS=(www ftp mail club)
- 方法二，为单个元素赋值：
 - 格式：**数组名[下标]=值**
 - 示例：FQDNS[0]=www.tarena.com
FQDNS[1]=mail.tarena.com
FQDNS[2]=club.tarena.com

下标从 0 开始



输出数组元素

知识讲解

- 获取单个数组元素
 - 格式：**\${数组名[下标]}**
- 获取所有数组元素
 - 格式：**\${数组名[@]}**
- 获取数组元素个数
 - 格式：**\${#数组名[@]}**
- 获取连续的多个数组元素
 - 格式：**\${数组名[@]:起始下标:元素个数}**

```
root@svr5:~  
[root@svr5 ~]# echo ${SVRS[1]}  
ftp  
[root@svr5 ~]# echo ${SVRS[@]}  
www ftp mail club  
[root@svr5 ~]# echo ${#SVRS[@]}  
4  
[root@svr5 ~]# echo ${SVRS[@]:1:2}  
ftp mail
```



expect预期交互

expect简介

- 基于TCL编写的自动交互式程序
 - 可以用在Shell脚本中，为交互式过程自动输送预先准备的文本或指令，而无需人工干预
 - 触发的依据是预期会出现的特征提示文本

```
[root@svr5 ~]# yum -y install expect
```

```
...
```

```
Installed:
```

```
expect.x86_64 0:5.44.1.15-5.el6_4
```

```
Dependency Installed:
```

```
tcl.x86_64 1:8.5.7-6.el6
```

expect应用示例

- 任务目标
 - 实现SSH自动登录，并远程执行指令

知识讲解

```
[root@svr5 ~]# vim ssh.sh
#!/bin/bash
host=192.168.4.5
expect << EOF
spawn ssh $host
expect "password"      {send "123456\r"}
expect "#"              {send "touch /a.txt\r"}
expect "#"              {send "exit\r"}
```



案例3：expect预期交互

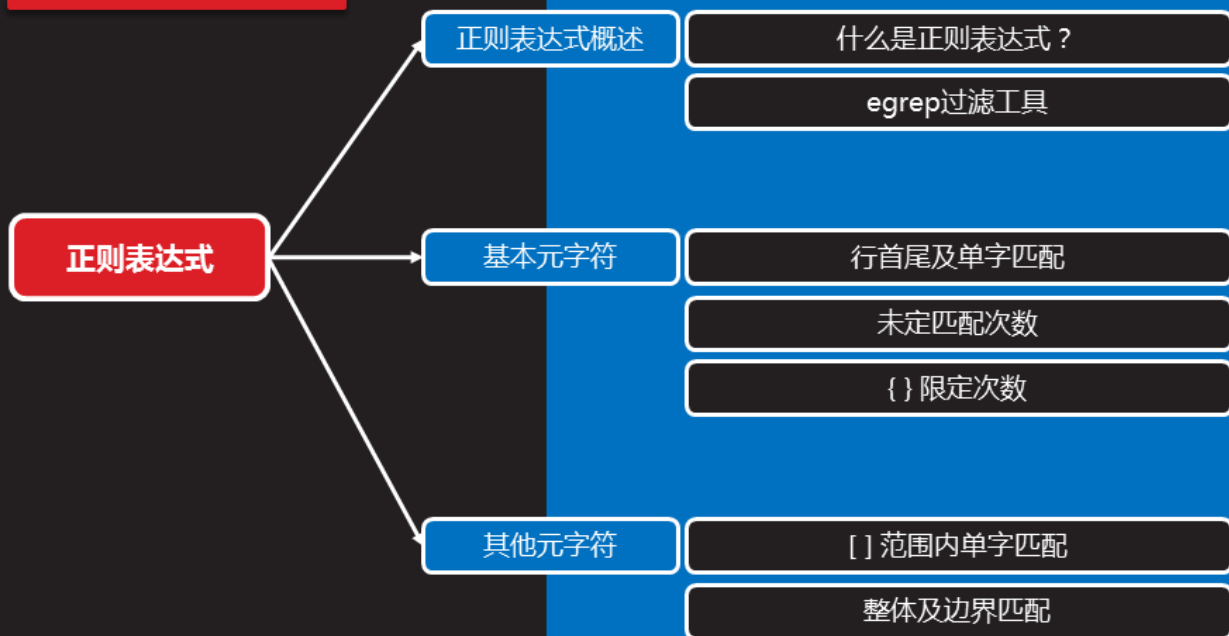
编写一个expect脚本，实现SSH登录的自动交互

- 目标主机地址为192.168.4.5
- 登入后在目标主机建立测试文件 /tmp/mike.txt

课堂练习



正则表达式



正则表达式概述

什么是正则表达式？

- Regular Express ?
 - 使用 “一串符号” 来描述有共同属性的数据

知识讲解



egrep过滤工具

- 文本处理顺序
 - 以行为单位，逐行进行处理
 - 默认只输出与表达式相匹配的文本行
- 基本用法
 - 格式1：**egrep** [选项] '正则表达式' 文件...
 - 格式2：**前置命令** | **egrep** [选项] '正则表达式'

知识讲解

等同于 `grep -E`，
表示允许使用扩展的正则表达式



egrep过滤工具（续1）

知识讲解

- 常用命令选项
 - -i：忽略字母大小写
 - -v：条件取反
 - -c：统计匹配的行数
 - -q：静默、无任何输出，一般用于检测
 - -n：显示出匹配结果所在的行号
 - --color：标红显示匹配字符串

—— 看 \$? 返回值，
如果为0，说明有匹配，否则无匹配



基本元字符

行首尾及单字匹配

知识讲解

类型	含义	示例	说明
^	匹配行首	^abc	以abc开头的行
		^#	以#号开头的行（比如注释行）
\$	匹配行尾	abc\$	以abc结尾的行
		^\$	空行
.	单个字符	.	除换行符（\n）以外的任意单个字符

```
[root@svr5 ~]# egrep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash           //找出root的账号记录
[root@svr5 ~]# egrep -c '/bin/bash$' /etc/passwd
23                                           //统计使用bash作登录Shell的用户数量
```



未定匹配次数

知识讲解

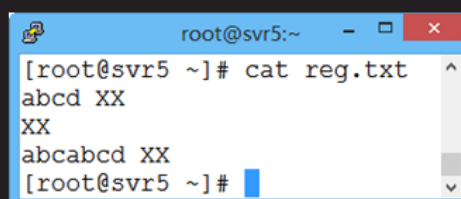
类型	含义	示例	说明
+	最少匹配一次	a+	一个或多个连续的 a
		(abc)+	一个或多个连续的 abc
?	最多匹配一次	a?	0个或1个 a
		(abc)?	0个或1个 abc
*	匹配任意次数	a*	0个或多个连续的 a
		(abc)*	0个或多个连续的 abc
		.*	任意长度的任意字符串



未定匹配次数（续1）

知识讲解

```
[root@svr5 ~]# egrep '(abc)+' reg.txt
abcd XX
abcabcd XX
[root@svr5 ~]# egrep '(abc)*' reg.txt
abcd XX
XX
abcabcd XX
```



```
root@svr5:~  
[root@svr5 ~]# cat reg.txt  
abcd XX  
XX  
abcabcd XX  
[root@svr5 ~]#
```



{ } 限定次数

- 限定表达式的匹配次数
 - {n}、{n,m}、{n,}

知识讲解

类型	含义	示例	说明
{n}	匹配n次	(ab){3}	匹配 ababab
{n,m}	匹配n-m次	(ab){1,3}	匹配 ab、abab、ababab
{n,}	匹配至少n次	(ab){2,}	匹配2个及以上连续的 ab



{ } 限定次数 (续1)

知识讲解

```
[root@svr5 ~]# egrep '(abc){2}' reg.txt
```

```
abcabcd XX
```

```
[root@svr5 ~]# egrep '(abc){1,}' reg.txt
```

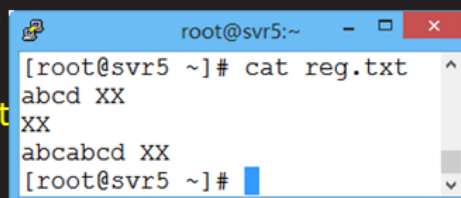
```
abcd XX
```

```
abcabcd XX
```

```
[root@svr5 ~]# egrep '(abc){1,3}' reg.txt
```

```
abcd XX
```

```
abcabcd XX
```



```
root@svr5:~  
[root@svr5 ~]# cat reg.txt  
abcd XX  
XX  
abcabcd XX  
[root@svr5 ~]#
```



其他元字符

[] 范围内单字匹配

- 匹配指定字符集合内的任何一个字符
 - []内加^可取反

知识讲解

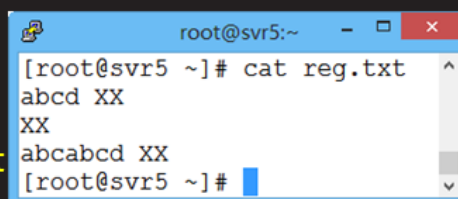
示 例	说 明
[alc45_?]	匹配 a、l、c、4、5、_、?
[a-z]	匹配任意小写字母
[A-Z]	匹配任意大写字母
[0-9]	匹配任意数字
[a-Z0-9]	匹配任意字母或数字
[^A-Z]	匹配包括非大写字母的行
^[^a-z]	匹配不以小写字母开头的行



[] 范围内单字匹配（续1）

知识讲解

```
[root@svr5 ~]# egrep '^[A-Z]' reg.txt
abcd XX
abcabcd XX
[root@svr5 ~]# egrep '^[^a-z]' reg.txt
XX
abcabcd XX
[root@svr5 ~]# egrep 'bc[dfx]' reg.txt
abcd XX
abcabcd XX
```



```
root@svr5:~  
[root@svr5 ~]# cat reg.txt  
abcd XX  
XX  
abcabcd XX  
[root@svr5 ~]#
```



整体及边界匹配

知识讲解

类 型	含 义	示 例	说 明
()	组合为整体	ab{1,3}	匹配 ab、abb、abbb
		(ab){1,3}	匹配 ab、abab、ababab
	或者	root bin	匹配 root、bin
\b	单词边界	\broot\b	匹配单词root，不匹配 keroot、rooty、brooty等字符串
\<	单词的开头	\<th	匹配以th开头的单词
\>	单词的结束	\<root\>	作用与 \broot\b 相同

\ 为转义符号，可以为一些普通字符赋予特殊含义，或者将一些特殊字符变为普通字符。



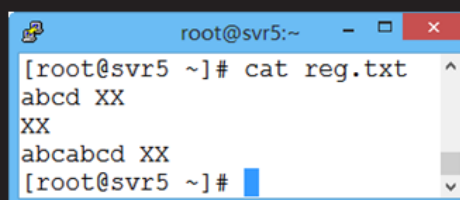
整体及边界匹配（续1）

知识讲解

```
[root@svr5 ~]# egrep '^root|^bin' /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
[root@svr5 ~]# egrep '\<abcd\>' reg.txt
abcd XX
```

```
[root@svr5 ~]# egrep 'abcd\>' reg.txt
abcd XX
abcabcd XX
```



```
root@svr5:~  
[root@svr5 ~]# cat reg.txt  
abcd XX  
XX  
abcabcd XX  
[root@svr5 ~]#
```



案例4：使用正则表达式

1. 利用egrep工具练习正则表达式的基本用法

课堂练习



总结和答疑

