

Shell脚本编程

NSD SHELL

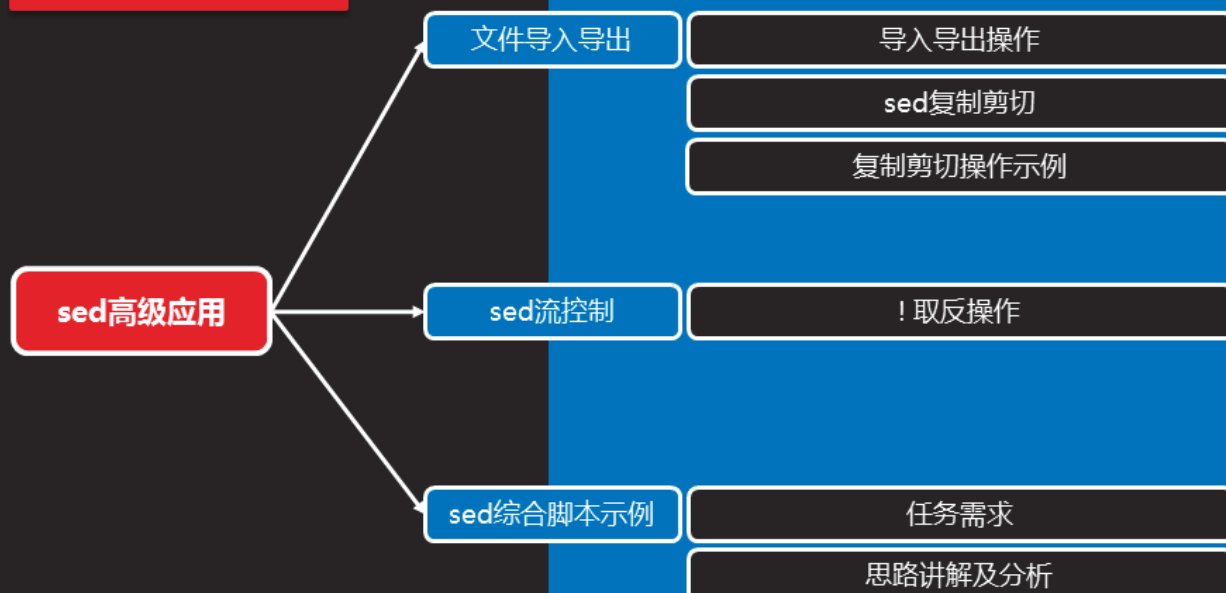
DAY05

内容

上午	09:00 ~ 09:30	作业讲解与回顾
	09:30 ~ 10:20	sed高级应用
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	awk基本用法
	15:00 ~ 15:50	
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



sed高级应用



文件导入导出

导入导出操作

- 基本动作
 - r 动作应结合 -i 选项才会存入，否则只输出
 - w 动作以覆盖的方式另存为新文件

知识讲解

操作符	用途	指令示例
r	读取文件	3r b.txt 在第3行下方插入文件b.txt
		4,7r b.txt 在第4-7每一行后插入文件b.txt
w	保存到文件	3w c.txt 将第3行另存为文件c.txt
		4,7w c.txt 将第4-7行另存为文件c.txt



导入导出操作（续1）

知识讲解

```
[root@svr5 ~]# sed '2r m.txt' reg.txt
```

```
abcd XX
```

```
XX
```

//在第2行下方插入m.txt

```
11111111 Tarena
```

```
222 IT Group
```

```
abcabcd XX
```

```
[root@svr5 ~]# sed '/^abca/r m.txt' reg.txt
```

```
abcd XX
```

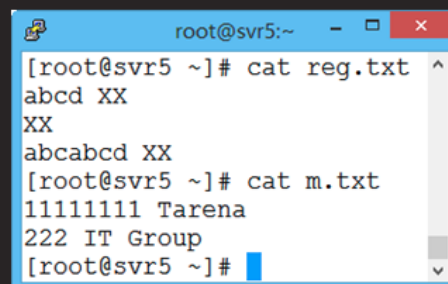
```
XX
```

```
abcabcd XX
```

//在以abca开头的行下方插入m.txt

```
11111111 Tarena
```

```
222 IT Group
```



```
root@svr5:~  
[root@svr5 ~]# cat reg.txt  
abcd XX  
XX  
abcabcd XX  
[root@svr5 ~]# cat m.txt  
11111111 Tarena  
222 IT Group  
[root@svr5 ~]#
```



导入导出操作（续2）

知识讲解

```
[root@svr5 ~]# sed -n '1,2w c.txt' reg.txt
```

```
[root@svr5 ~]# cat c.txt
```

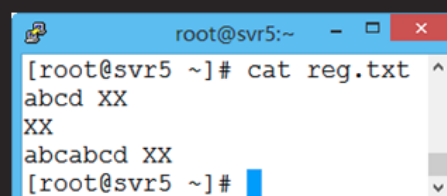
```
abcd XX
```

```
XX
```

```
[root@svr5 ~]# sed -n '/^XX/w d.txt' reg.txt
```

```
[root@svr5 ~]# cat d.txt
```

```
XX
```



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

与以下操作效果相同：sed -n '/^XX/p' reg.txt > d.txt



sed流控制

！取反操作

- 用途
 - 根据定址条件取反
- 示例
 - 列出不使用bash的用户账号记录

```
[root@svr5 ~]# sed -n '/bash$/!p' /etc/passwd  
bin:x:1:1:bin:/bin:/sbin/nologin  
daemon:x:2:2:daemon:/sbin:/sbin/nologin  
adm:x:3:4:adm:/var/adm:/sbin/nologin  
...
```

sed综合脚本示例

任务需求

- 根据/etc/passwd提取密码串
 - 找到使用bash作登录Shell的本地用户
 - 列出这些用户的shadow密码记录
 - 按每行“用户名 --> 密码记录” 保存结果

知识讲解

```
root@svr5:~  
[root@svr5 ~]# head -2 /tmp/getupwd.log  
root --> $1$vEpH83MN$nlajDFq5SialdzyJQyWs3/  
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0
```



思路讲解及分析

参考思路、分析

知识讲解

```
root@svr5:~  
[root@svr5 ~]# cat getupwd.sh  
#!/bin/bash  
> /tmp/getupwd.log  
sed -n '/:\/bin\/bash$/w /tmp/urec.tmp' /etc/passwd  
UNUM=$(egrep -c '.' /tmp/urec.tmp)  
while [ ${i:=1} -le $UNUM ]  
do  
    UREC=$(sed -n "${i}p" /tmp/urec.tmp)  
    NAME=${UREC%%:*}  
    PREC=$(sed -n "/^$NAME:/p" /etc/shadow)  
    PASS=${PREC#*:  
    PASS=${PASS%:*}  
    echo "$NAME --> $PASS" >> /tmp/getupwd.log  
    let i++  
done  
/bin/rm -rf /tmp/urec.tmp  
echo "用户分析完毕，请查阅文件 /tmp/getupwd.log"  
[root@svr5 ~]#  
[root@svr5 ~]# head -2 /tmp/getupwd.log  
root --> $1$vEpH83MNSnlaJDFq5SialdzyJQyWs3/  
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0  
[root@svr5 ~]#
```

案例1：sed综合脚本应用

编写脚本getupwd.sh，实现以下需求：

- 1) 找到使用bash作登录Shell的本地用户
- 2) 列出这些用户的shadow密码记录
- 3) 按每行“用户名 --> 密码记录”保存到getupwd.log

课堂练习

awk基本用法

awk基本用法

awk命令解析

awk工具概述

命令格式解析

awk内置变量

awk过滤的时机

awk处理条件

处理条件概述

条件设置示例

多个条件的组合

变量的运算

awk综合脚本示例

任务需求

思路讲解及分析

awk命令解析

awk工具概述

知识讲解

- awk编程语言/数据处理引擎
 - 创造者：Aho、Weinberger、Kernighan
 - 基于模式匹配检查输入文本，逐行处理并输出
 - 通常用在Shell脚本中，获取指定的数据
 - 单独用时，可对文本数据做统计



命令格式解析

知识讲解

- 主要用法
 - 格式1：前置命令 | awk [选项] '[条件]{指令}'
 - 格式2：awk [选项] '[条件]{指令}' 文件.. ..

多条语句可以分号分隔

print 是最常用的指令

```
[root@svr5 ~]# cat test.txt
```

```
hello the world  
welcome to beijing
```

```
[root@svr5 ~]# awk '{print $1,$3}' test.txt
```

```
hello world  
welcome beijing
```



命令格式解析（续1）

知识讲解

- 常用命令选项
 - -F：指定分隔符，可省略（默认空格或Tab位）

```
[root@svr5 ~]# awk -F: '{print $1,$3}' /etc/passwd
root 0
bin 1
daemon 2
adm 3
...
```



命令格式解析（续2）

知识讲解

- 检查登录失败的IP地址有哪些

```
[root@svr5 ~]# awk '/Failed/{print $11}' /var/log/secure
192.168.2.254
192.168.2.100
...
```

- 检查内存的剩余容量

```
[root@svr5 ~]# free | awk '/Mem/{print $4}'
```

- 过滤网络流量

```
[root@svr5 ~]# ifconfig eth0 | awk '/RX p/{print $5}'
```



awk内置变量

- 有特殊含义，可直接使用

知识讲解

变 量	用 途
FS	保存或设置字段分隔符，例如 FS= ":", 与-F功能一样
\$n	指定分隔的第n个字段，如\$1、\$3分别表示第1、第3列
\$0	当前读入的整行文本内容
NF	记录当前处理行的字段个数（列数）
NR	记录当前已读入行的数量（行数）



awk内置变量（续1）

```
[root@svr5 ~]# awk -F: '{print NR,NF}' /etc/passwd
```

知识讲解

```
[root@svr5 ~]# awk -F: '{print $NF}' /etc/passwd
```

└── 输出每行最后一个字段

```
[root@svr5 ~]# awk -F: '{print "用户名:",$1,"解释器:",$7}' /etc/passwd
用户名: root 解释器: /bin/bash
用户名: bin 解释器: /sbin/nologin
... ..
```



awk过滤的时机

知识讲解

- 在所有行前处理，**BEGIN{ }**
 - 读入第一行文本之前执行
 - 一般用来初始化操作
- 逐行处理，**{ }**
 - 逐行读入文本执行相应的处理
 - 是最常见的编辑指令块
- 在所有行后处理，**END{ }**
 - 处理完最后一行文本之后执行
 - 一般用来输出处理结果

可单独使用，
也可以同时一起使用



awk过滤的时机（续1）

知识讲解

```
[root@svr5 ~]# awk 'BEGIN {a=34;print a+12}'  
46                                     //预处理不需要数据文件
```

```
[root@svr5 ~]# awk 'BEGIN{x=0}/<bash$/ {x++}\nEND{print x}' /etc/passwd  
59                                     //统计使用bash的用户个数
```

```
[root@svr5 ~]# awk 'BEGIN {print NR} END{print NR}' m.txt  
0                                     //预处理时，行数为0  
2                                     //全部处理完以后，行数为已读入文本的行数
```



案例2：使用awk提取文本

课堂练习

1. 练习awk工具的基本用法
2. 提取本机的网卡流量、根分区剩余容量，获取远程失败的IP地址
3. 格式化输出/etc/passwd文件：
 - 1) 只显示用户名、UID、宿主目录3列
 - 2) 给每列加标题，最后输出处理的总行数

```
User      UID      Home
root      0        /root
bin       1        /bin
daemon    2        /sbin
adm       3        /var/adm
.. ..
Total 59 lines.
```



awk处理条件

处理条件概述

知识讲解

- 所有的行全部处理并输出吗？
- 怎么限制处理的条件？
- 根据多个条件来处理指定的行？



处理条件概述（续1）

知识讲解

- 格式回顾
 - awk [选项] '[条件]{编辑指令}' 文件.. ..
- 条件的表现形式？
 - 正则表达式
 - 数值/字符串比较
 - 逻辑比较
 - 运算符



条件设置示例

知识讲解

- 正则表达式
 - /正则表达式/
 - ~ 匹配、!~ 不匹配

```
[root@svr5 ~]# awk -F: '/^ro/{print}' /etc/passwd
root:x:0:0:root:/root:/bin/bash //列出以ro开头的用户记录
```

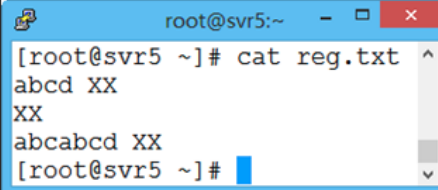
```
[root@svr5 ~]# awk -F: '$7!~/bash$/{print $1,$7}' /etc/passwd
bin /sbin/nologin //列出第7个字段不以bash结
daemon /sbin/nologin //尾的用户名、登录Shell
```



条件设置示例（续1）

知识讲解

- 数值比较
 - == 等于、!= 不等于
 - > 大于、>= 大于或等于
 - < 小于、<= 小于或等于



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

```
[root@svr5 ~]# awk 'NR==2{print}' reg.txt
XX //输出第2行文本
```

```
[root@svr5 ~]# awk '$2!="XX"{print}' reg.txt
XX //输出第2列不是XX的行
```

```
[root@svr5 ~]# awk 'NF>=2{print}' reg.txt
abcd XX //输出包含2个及以上字段的行
abcbcd XX
```



多个条件的组合

知识讲解

- 逻辑比较测试
 - && 逻辑与：期望多个条件都成立
 - || 逻辑或：只要有一个条件成立即满足要求

```
[root@svr5 ~]# awk -F: '$3>=0&&$3<2{print $1,$3}' /etc/passwd
//列出UID小于2的用户信息
```

```
root 0
```

```
bin 1
```

```
[root@svr5 ~]# awk -F: '$3==1 || $3==7{print $1,$3}' /etc/passwd
//列出UID为1或7的用户信息
```

```
bin 1
```

```
halt 7
```



变量的运算

知识讲解

- 运算符
 - +、-、*、/、%
 - ++、--、+=、-=、*=、/=

```
[root@svr5 ~]# awk 'NR%2==1{print}' reg.txt
abcd XX
//输出奇数行文本
```

```
abcbcd XX
```

```
[root@svr5 ~]# awk 'BEGIN{i=0} {i+=NF} END{print i}' reg.txt
5
//统计文本的总字段个数
```

```
[root@svr5 ~]# seq 200 | awk 'BEGIN{i=0} ($0%3==0)&& \
($0%13==0){i++} END{print i}'
5
//计算能同时被3和13整除的整数个数
```



案例3：awk处理条件

课堂
练习

1. 列出UID间于1~1000的用户详细信息
2. 输出/etc/hosts文件内以127或192开头的记录
3. 列出100以内整数中7的倍数或是含7的数



awk综合脚本示例

任务需求

知识讲解

- 根据/etc/passwd提取密码串
 - 找到使用bash作登录Shell的本地用户
 - 列出这些用户的shadow密码记录
 - 按每行“用户名 --> 密码记录”保存结果

```
root --> $1$vEpH83MN$n1aJDFq5SialdzyJQyWs3/  
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0
```



思路讲解及分析

参考思路、分析

知识讲解

```
root@svr5:~  
[root@svr5 ~]# cat getupwd-awk.sh  
#!/bin/bash  
## 创建空文件  
> /tmp/getupwd.log  
## 提取用户名列表  
awk -F: '/:\/bin\/bash${print $1}' /etc/passwd > /tmp/users.tmp  
## 通过for循环遍历用户名、查询密码记录，保存结果  
for NAME in $(cat /tmp/users.tmp)  
do  
    grep "^$NAME:" /etc/shadow | awk -F: '{print $1" --> "$2 | \  
    "cat >> /tmp/getupwd.log"}'  
done  
echo "用户分析完毕，请查阅文件 /tmp/getupwd.log" ## 完成后提示  
[root@svr5 ~]#  
[root@svr5 ~]# head -2 /tmp/getupwd.log  
root --> $1$vEpH83MN$n1aJDFq5SialdzyJQyWs3/  
nick --> $1$ZrLQZB3g$phYBuzHeU9YiqgFQWZZaf0  
[root@svr5 ~]#
```

双引号调用外部Shell命令



案例4：awk综合脚本应用

课堂练习

编写脚本getupwd-awk.sh，实现以下需求：

- 1) 找到使用bash作登录Shell的本地用户
- 2) 列出这些用户的shadow密码记录
- 3) 按每行“用户名 --> 密码记录”保存到getupwd.log



总结和答疑



sed替换

问题现象

- 故障错误信息

```
[root@svr5 ~]# sed "s/a/b" tesh.txt  
sed: -e expression #1, char 5: unterminated `s' command
```

原因分析

知识讲解

- 分析故障
 - 报错信息：unterminated `s' command
- 分析故障原因
 - s替换没找到合法的结束符
 - s/旧/新/，使用s替换时，最后需要使用/结尾



awk分隔符

问题现象

- 故障错误信息

```
[root@svr5 ~]# awk '{print $1,$3}' /etc/passwd | head -2  
root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin
```

知识讲解



原因分析

- 分析故障
 - 报错信息：输出的信息不是希望的数据
- 分析故障原因
 - 默认awk分隔符为空格和tab键
 - 通过-F指定分隔符

知识讲解



