**shell day 04**

================================================

**使用循环嵌套，得到50个8位长度的随机字符存在pass文件中**

#!/bin/bash

x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

for j in {1..500}

do

for i in {1..18}

do

a=$[RANDOM%62]

b=${x:a:1}

pass=$pass$b

done

echo $pass >> pass

pass=

done

-----------------------------------------------------------

**2，字符串的替换**

**${变量名/被替换内容/新内容}**

a=1234 //定义素材

echo ${a/3/6} //把3换成6

a=11223344 //定义素材

echo ${a/3/6} //把3换成6,默认只换一个

echo ${a//3/6} //使用两个/可以实现把所有3换成6

echo ${a/3//} //把3换成空，相当于删除

a=111222 //定义素材

echo ${a/2/6} //替换1个2

echo ${a//2/6} //替换所有2

echo ${a/22/66} //替换前两个2

echo ${a/222/266} //根据需求替换

echo $a //注意，变量只要不重新赋值，原始内容不会变

a=666 //定义素材

a=${a/6/8} //替换变量内容后重新赋值

echo $a //变量内容发生改变

**3，字符串的删除**

${变量名#被删除的内容} 从左往右删除 掐头

${变量名%被删除的内容} 从右往左删除 去尾

a=abcdefg

echo ${a#abc} //从左往右删除到c

echo ${a#\*c} //效果同上，更精简

echo ${a#abcde} //从左往右删除到e

echo ${a#\*e} //效果同上，更精简

a=abcxyzabcxyz

echo ${a#\*b} //从左往右删除到第一个b

echo ${a##\*b} //从左往右删除到最后一个b

a=abcdef

echo ${a%def} //从右往左删除def

echo ${a%d\*} //效果同上，更精简

a=abcdefabcdef

echo ${a%%d\*} //从右往左删除到最后一个d

**练习：**

**编写脚本，实现批量修改文件扩展名**

touch abc{01..10}.txt //创建10个文件作为素材

#!/bin/bash

for i in $(ls \*.txt) //找到所有txt文件交给for循环

do

n=${i%.\*} //先用去尾删除扩展名,n存储了纯粹的文件名

mv $i $n.doc //再将原文件修改为doc扩展名

done

改良版

#!/bin/bash

for i in $(ls \*.$1) //使用位置变量$1表示要改的扩展名

do

n=${i%.\*}

mv $i $n.$2 //使用位置变量$2表示要改成啥

done

**变量初值（备用值）的定义**

${变量名:-初值} //如果变量有值则使用本身的值，如果变

量为空，则使用初值

编写脚本，可以创建用户与配置密码，密码可以自定义，也

可以使用默认的123456

#!/bin/bash

read -p "请输入用户名：" u

useradd $u

read -p "请输入密码：" p

echo ${p:-123456} | passwd --stdin $n //当用户没有输入密码时

密码就是123456

**总结：**

${变量名:} 截取

${变量名/} 替换

${变量名#} 掐头

${变量名%} 去尾

${变量名:-} 定义初值

-----------------------------------------------------------------------------------

**正则表达式，使用若干符号配合某工具对文档实现过滤、查找**

**、修改等功能**

head -5 /etc/passwd > user //准备素材

grep bin user //找有bin的行

grep ^bin user //找以bin开头的行

grep bash user //找有bash的行

grep bash$ user //找以bash结尾的行

vim user //编辑文档添加空行

grep -n ^$ user //找空行，加n选项可以显示行号

grep "[root]" user //找root四个字符任意一个

grep "[rot]" user //效果同上，找rot任意一个字符

grep "rot" user //找连续的rot字符串

grep "[a-z]" user //找所有小写字母

grep "[A-Z]" user //找所有大写字母

grep "[a-Z]" user //找所有字母

grep "[0-9]" user //找所有数字

grep "[^a-Z]" user //找字母之外的内容，^写在[]里是取反效果

grep "r..t" user //找rt之间有2个任意字符的行

grep "r...t" user //找rt之间有3个任意字符的行，没有匹配内容，就无输出

grep "\*" user //错误用法，\*号是匹配前一个字符任意次，不能单独使用

grep "ro\*t" user //找rt，中间的o有没有都行，有几次都行

grep "." user //找任意单个字符，文档中每个字符都可以理解为任意字符

grep ".\*" user //找任意，包括空行 .与\*的组合在正则中相当于通配符的效果

grep "ro\{2,4\}t" user //找rt，中间的o可以是2~4个

grep "ro\{2,\}t" user //找rt，中间的o可以是2个以及2个以

上

grep "ro\{3,7\}t" user //找rt，中间的o可以是3~7个，没有匹

配条件就没有任何显示

grep "ro\{3,\}t" user //找rt，中间的o可以是3个以及3个以

上

grep "ro\{2\}t" user //找rt，中间的o必须是2个

grep "o\{2,3\}" user //找o出现2~3次

grep -E "o{2,3}" user //还可以使用-E选项，支持扩展正则，比较精简

egrep "o{2,3}" user //还可以更换egrep命令，也支持扩展正则

grep "\(0:\)\{2\}" user //找连续2个0:

grep -E "(0:){2}" user //找连续2个0: 使用扩展正则更精简

grep "ro\{1,\}t" user //找1次以及1次以上的o

egrep "ro{1,}t" user //效果同上,egrep支持扩展正则，或者grep -E

egrep "ro+t" user //效果同上，最精简

grep "roo\{0,1\}t" user //第二个o要出现0~1次

egrep "roo{0,1}t" user //效果同上

egrep "roo?t" user //效果同上，最精简

grep "ro\{2\}t" user //找o出现2次的

egrep "ro{2}t" user //效果同上

egrep "^root|^bin" user //找root或者以bin开头的行

egrep "^(root|bin)" user //效果同上

[root@svr7 opt]# cat abc

the

there

123the

the\_

abcthe\_

#the

egrep "\bthe\b" abc //找the，前后不允许出现数

字，字母，下划线

**sed 流式编辑器，可以对文档进行非交互式增删改查，逐行处理**

**使用方式：**

**1，前置指令 | sed 选项 定址符 指令**

**2，sed 选项 定址符 指令 被处理的文档**

**选项： -n 屏蔽默认输出 -r 支持扩展正则 -i 写入文件**

**指令：p 输出 d 删除 s替换**

sed -n '1p' user //输出第1行

sed -n '2p' user //输出第2行

sed -n '3p' user //输出第3行

sed -n '2,4p' user //输出第2~4行

sed -n '2,+1p' user //输出第2行以及后面1行

sed -n '2p;4p' user //输出第2行，第4行

sed -n '/^root/p' user //在sed中使用正则表达式输出以root开

头的行，匹配的内容要写在两个斜杠中间

grep "^root" user //效果同上

egrep "^root|^bin" user //找root或者bin开头的行

sed -nr '/^root|^bin/p' user //在sed中查找，-r使用扩展正

则，效果同上

sed -n '=' user //查看所有行号

sed -n '$=' user //查看最后一行的行号，相当于查看文档

总共有几行

sed -n '$p' user //输出最后一行的内容

sed '1d' user //删第1行

sed 'd' user //删所有

sed '3d' user //删第3行

sed '2,5d' user //删2~5行

sed '4,+2d' user //删4行以及后面2行

sed '1d;3d' user //删第1行，第3行

sed '$d' user //删除最后一行

sed -n '$p' user //查看最后一行

sed -n '1!p' user //查看除了第1行以外的行, !代表取反

sed '/root/d' user //删除含有root的行

sed '1!d' user //删掉除第1行以外的行

regular.pdf 正则表达式练习题

regular\_express.txt 正则表达式练习素材

练习：

1 简述grep工具的-q选项的含义（egrep同样适用）。

2 正则表达式中的+、？、\*分别表示什么含义？

3 如何编写正则表达式匹配11位的手机号？

4 简述sed定址符的作用及表示方式。

参考答案：

1 简述grep工具的-q选项的含义（egrep同样适用）。

选项-q的作用是静默、无任何输出，效果类似于正常的grep操作添加了&> /dev/null来屏蔽输出

2 正则表达式中的+、？、\*分别表示什么含义？

这三个字符用来限制前面的关键词的匹配次数，含义分别如下：

• +：最少匹配一次，比如a+可匹配a、aa、aaa等

• ？：最多匹配一次，比如a?可匹配零个或一个a

• \*：匹配任意多次，比如a\*可匹配零个或任意多个连续的a

3 如何编写正则表达式匹配11位的手机号？

准备测试文件：

1. [root@svr5 ~]# cat tel.txt

2. 01012315

3. 137012345678

4. 13401234567

5. 10086

6. 18966677788

提取包含11位手机号的行：

1. [root@svr5 ~]# egrep '^1[0-9]{10}$' tel.txt

2. 13401234567

3. 18966677788

4 简述sed定址符的作用及表示方式。

作用：定址符（执行指令的条件）控制sed需要处理文本的范围；不加则逐行处理所有行

表示方式：定址符可以使用行号或正则表达式