

基础实验篇第1章

Shell编程： 批量文件重命名

1

Shell编程概述

2

Shell编程基础

3

Shell脚本

4

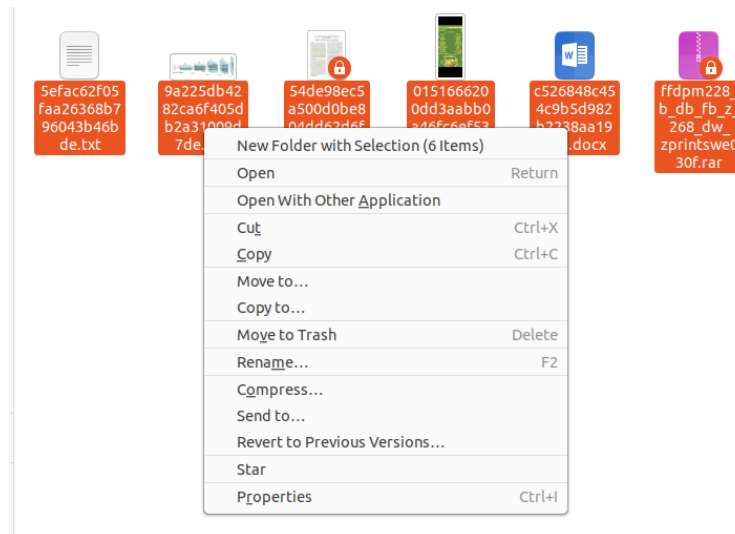
Shell实战：批量文件重命名

5

Shell扩展阅读



方法1:



方法2:

shell编程

Shell编程概述

什么是shell?

Shell是计算机系统提供的一种**命令行界面**，它允许**用户与操作系统**进行**交互**。通过shell，我们可以在命令行中输入各种命令来执行操作。

在UNIX和类UNIX操作系统中，常见的shell有Bourne Shell (sh)，Bourne Again Shell (bash) 和C Shell (csh)。在Windows系统中，常见的shell是命令提示符 (cmd.exe) 和PowerShell。

什么是shell编程?

Shell编程是一种**编写和执行命令**的方法，以便**自动化**完成各种任务。它使用的是一种**脚本语言**，可以在操作系统的命令行界面 (shell) 中运行。

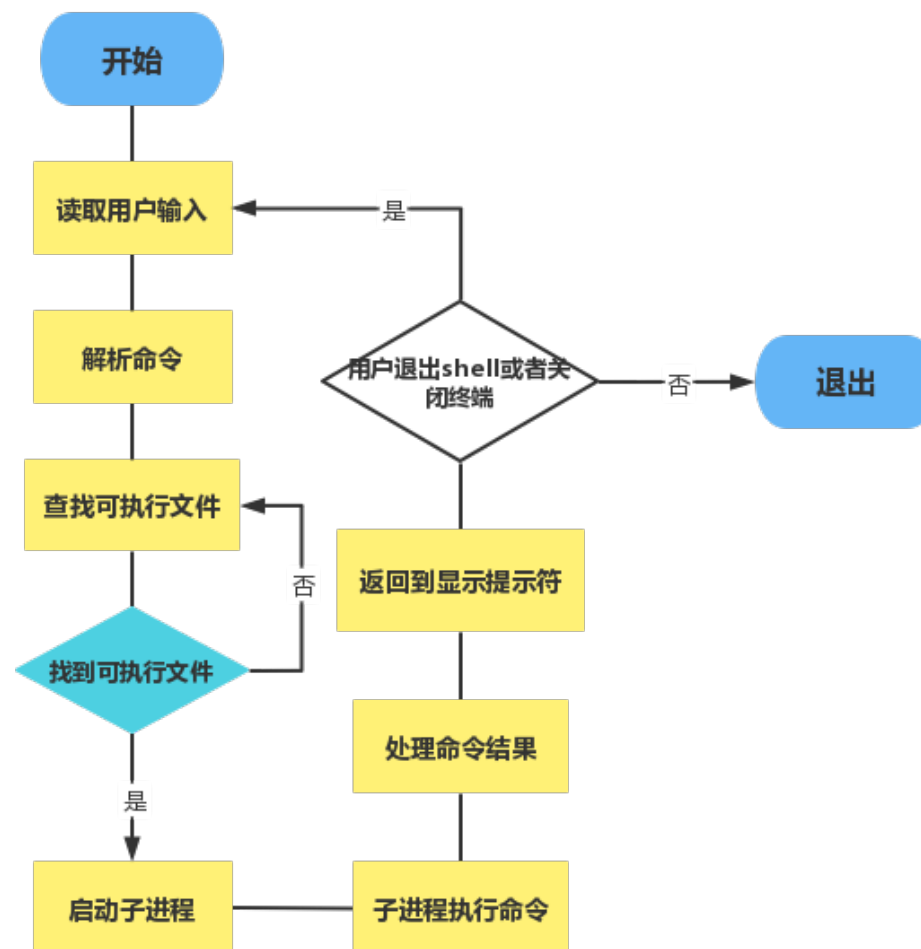
Shell编程的基本概念是将一系列**命令和逻辑**组织在一个**文本文件**中，称为脚本。脚本文件包含一系列指令，这些指令可以是操作系统命令、应用程序命令或自定义命令，用于执行特定的操作。

shell的原理是什么？

```

1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/wait.h>
4  #include <stdlib.h>
5
6  char command[256]
7  void main()
8  {
9      pid_t id=fork();
10     int rtn;//子进程的返回数值
11     while(1){//从终端读取要执行的命令
12         printf(">");
13         fgets(command,256,stdin);
14         command[strlen(command)-1]=0;
15         if(id==0){//子进程执行此命令
16             execlp(command,NULL);
17             //如果exec函数返回,表明没有正常执行命令,打印错误信息
18             perror(command);
19             exit(errno);
20         }
21         else{//父进程,等待子进程结束,并打印子进程的返回值
22             wait(&rtn);
23             printf("child process return %d\n",rtn);
24         }
25     }
26 }
    
```

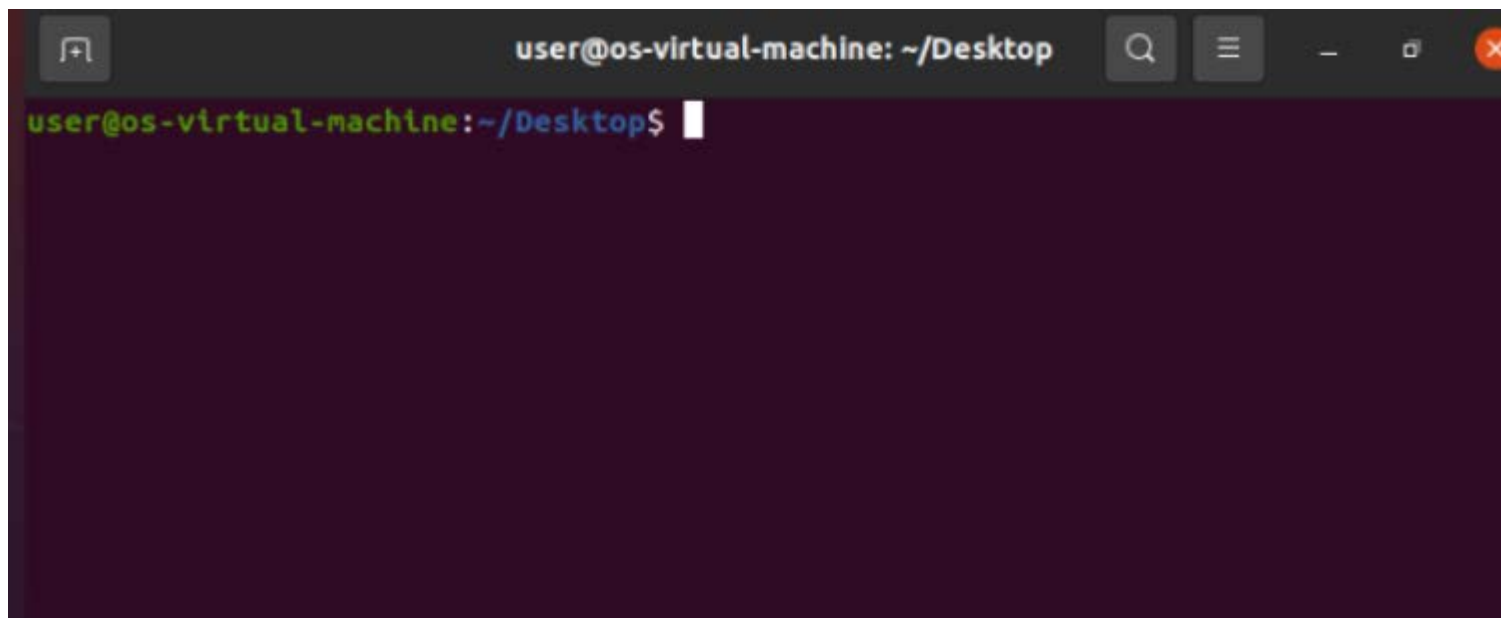
C语言编写的简单shell



Shell解释程序的执行过程

如何进入shell?

右键打开终端，就可以输入shell命令了。\$是命令提示符，可以在它后面输入命令。

A screenshot of a terminal window. The title bar at the top reads "user@os-virtual-machine: ~/Desktop". The terminal content shows the prompt "user@os-virtual-machine:~/Desktop\$" followed by a white cursor. The window has standard Linux window controls (minimize, maximize, close) on the right side.

```
user@os-virtual-machine: ~/Desktop
user@os-virtual-machine:~/Desktop$
```

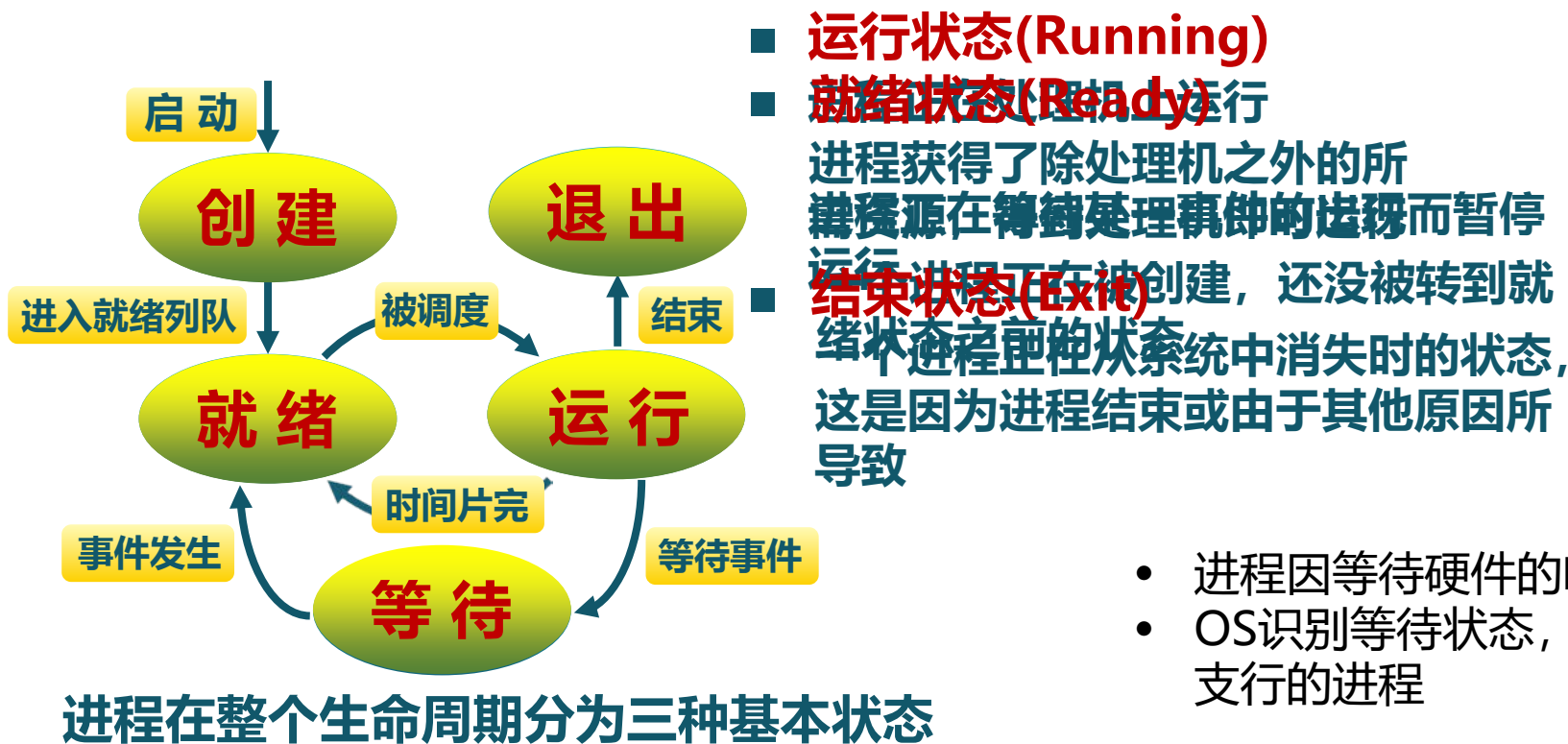
Shell与进程管理

从shell查看进程的状态

Demo:

使用命令查看进程状态: top, ps

目标程序: ping, while(1)形成的busywait



Shell与权限管理

从shell看多重权限管理

x86 特权级 – 简介

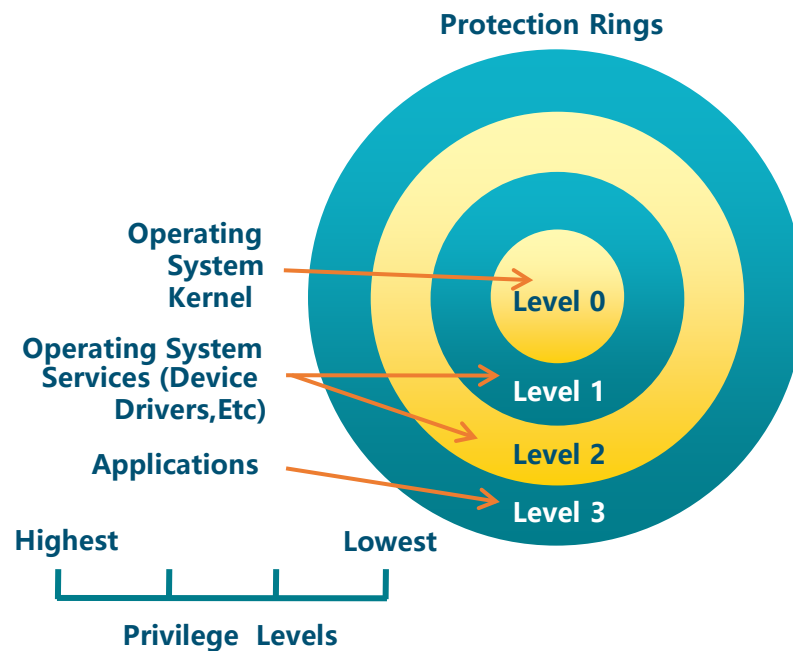


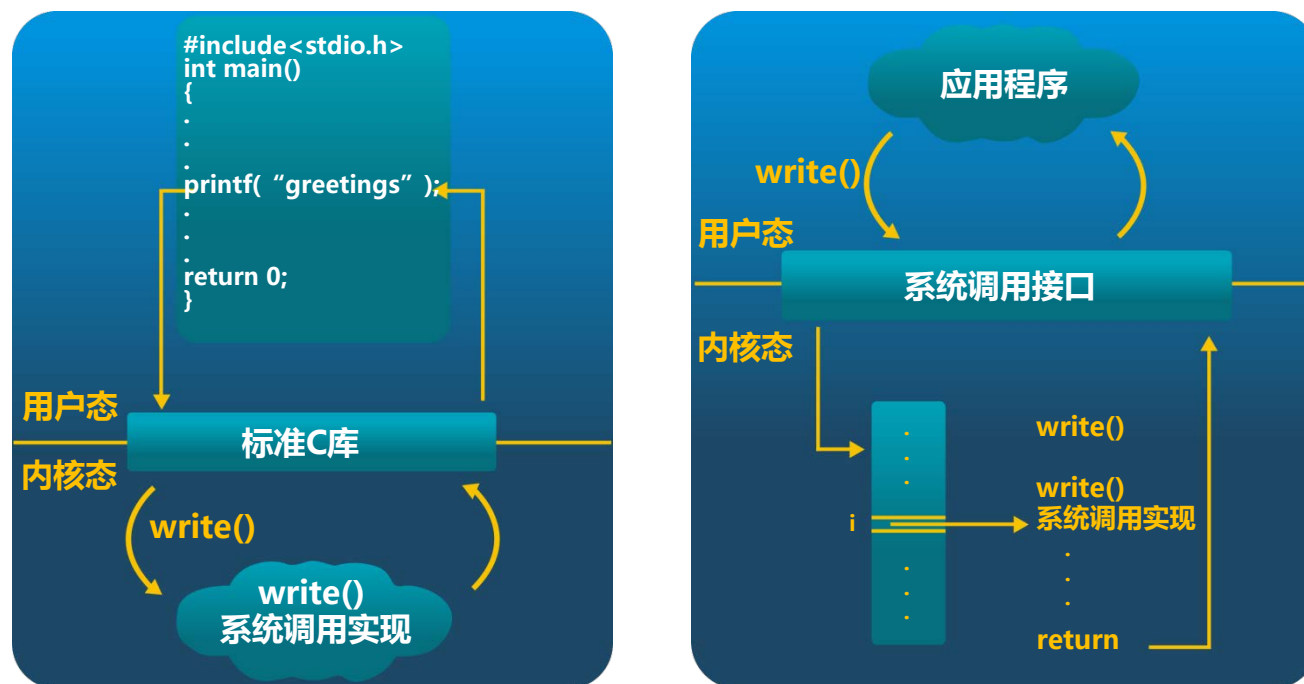
图 Protection Rings

- Linux 和 许多其它OS 只使用 ring 0 and ring 3

从shell看多重权限管理

标准C库的例子

- 应用程序调用printf() 时，会触发系统调用write()。



操作系统的用户，只有通过系统提供的命令或者API，才能执行特权指令

从shell看多重权限管理

设置文件和目录的权限chmod

■ chmod命令

u、g、o、a 分别表示
属主、属组、其他用户、所有用户

r、w、x 分别表示
读、写、运行权限

chmod [u~~g~~oa] [+~~-~~=] [rwx] 文件或目录...

或

+、-、= 分别表示
增加、去除、设置权限

chmod nnn 文件或目录...

3位八进制数

用户仅能访问
和执行管理员
允许的文件

■ 常用选项

- -R: 递归修改指定目录下所有子项的权限

Shell编程基础

shell的输入输出

1、输入

(1) 标准输入（如键盘输入）：一般用read命令，常用的有：

-a array	把读取的数据赋值给数组 array，从下标0开始；
-d delimiter	用字符串 delimiter 指定读取结束的位置，而不是一个换行符（读取到的数据不包括 delimiter）
-e	在获取用户输入的时候，对功能键进行编码转换，不会直接显示功能键对应的字符
-n num	读取 num 个字符，而不是整行字符
-p prompt	显示提示信息，提示内容为 prompt
-r	原样读取（Raw mode），不把反斜杠字符解释为转义字符
-s	不会在屏幕上显示输入的字符。一般用于输入密码和其它确认信息的时候
-t seconds:	设置超时时间，单位为秒。如果用户没有在指定时间内输入完成，那么 read 将会返回一个非 0 的退出状态，表示读取失败
-u fd	使用文件描述符fd作为输入源，而不是标准输入，类似于重定向

(2) 重定向输入：重新指定设备来代替键盘作为新的输入设备，这个设备一般指的是文件或命令的执行结果。

shell的输入输出

2、输出

(1) 标准输出（如终端显示）：

①echo命令：是shell中非常常用的命令，它的功能是打印任意字符到标准输出；

②printf命令：可以定义数据输出的格式，默认的 printf 不会像echo自动添加换行符，我们可以手动添加\n。

```
user@os-virtual-machine:~/Desktop$ test=os
user@os-virtual-machine:~/Desktop$ echo $test
os
user@os-virtual-machine:~/Desktop$ printf $test
osuser@os-virtual-machine:~/Desktop$ printf "$test \n"
os
user@os-virtual-machine:~/Desktop$
```

(2) 重定向输出：重新指定设备来代替显示器作为新的输出设备，而这个设备一般指的是文件。

shell的变量

1、定义变量：

变量名=变量值

值得注意的是：

- ①变量名以字母或下划线开头，区分大小写，建议全大写，不能以数字、特殊符号开头；
- ②赋值时“=”两边不能有空格。

对变量进行赋值的时候，可能会用到引号，主要有三种：

- ①双引号" "：允许通过\$符号引用其他变量值；
- ②单引号' '：禁止引用其他变量值，\$视为普通字符；
- ③反引号` `：命令替换，提取命令执行后的输出结果，其作用和\$(...)相同

从键盘输入内容来为变量赋值，可以用read命令：

read -p 提示信息 变量名

```
user@os-virtual-machine:~/Desktop$ read -p "请输入姓名：" NAME
请输入姓名：user
user@os-virtual-machine:~/Desktop$ echo $NAME
user
```

shell的变量

2、查看变量：

echo \$变量名

```
user@os-virtual-machine:~/Desktop$ PRODUCT=shell
user@os-virtual-machine:~/Desktop$ echo $PRODUCT2.0
.0
user@os-virtual-machine:~/Desktop$ echo $PRODUCT 2.0
shell 2.0
user@os-virtual-machine:~/Desktop$ echo "$PRODUCT 2.0"
shell 2.0
user@os-virtual-machine:~/Desktop$ echo "$PRODUCT"2.0
shell2.0
user@os-virtual-machine:~/Desktop$ echo "$PRODUCT2.0"
.0
user@os-virtual-machine:~/Desktop$ echo '$PRODUCT'2.0
$PRODUCT2.0
user@os-virtual-machine:~/Desktop$ test=`echo "$PRODUCT"2.0`
user@os-virtual-machine:~/Desktop$ echo test
test
user@os-virtual-machine:~/Desktop$ echo $test
shell2.0
```

定义变量

PRODUCT2没被定义所以无法输出

双引号允许通过\$符号引用其他变量值

单引号禁止引用其他变量值，\$视为普通字符

反撇号可直接调用其中的命令结果

查看变量值

shell的变量

3、变量的作用范围：默认情况下，新定义的变量只在当前的Shell环境中有效，因此被称为局部变量。当进入子程序或新的子Shell环境时，局部变量将无法再使用。可以通过内部命令`export`将指定的变量导出为全局变量，使用户定义的变量在所有的子Shell环境中能够继续使用：

格式1: `export 变量名`

格式2: `export 变量名=变量值`

shell如何进行文件重命名？

一般采用mv命令，用于单个文件的修改：

```
$mv fileA fileB
```

例如：

```
$mv test.txt os.txt
```

是将原名为test.txt的文件改名为os.txt

shell的循环与通配符

1、for循环：读取不同的变量值，用来逐个执行同一组命令

```
1  for 变量名 in 取值列表
2  do
3  命令序列 (命令行)
4  done
```

2、while循环：重复测试某个条件，只要条件不成立则反复执行

```
1  while 条件测试操作
2  do
3  命令序列
4  done
```

3、until语句：重复测试某个条件，只要条件不成立则反复执行

```
1  until 条件测试操作
2  do
3  命令序列
4  done
```

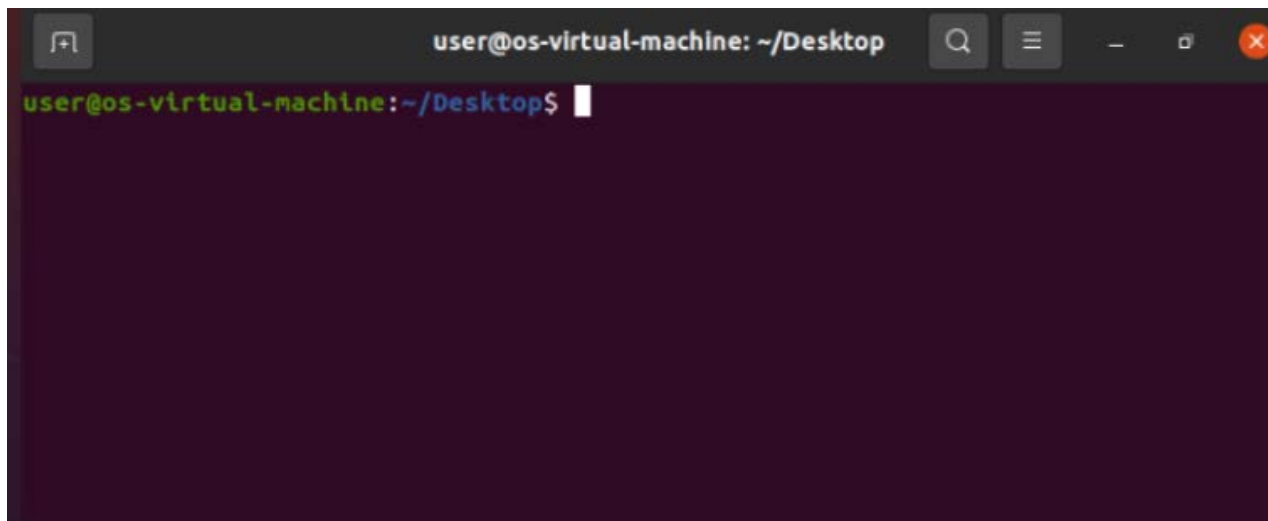
shell的循环与通配符

通配符：一种特殊语句，用来模糊搜索文件。当查找文件夹时，可以使用它来代替一个或多个真正字符；当不知道真正字符或者完整名字过长时，通常可以使用通配符来代替一个或多个真正的字符。

*： 匹配0个或多个字符	如a*c表示 a与c之间可以有任意长度的任意字符, 也可以一个也没有, 如aabcc、axyzc、 a012c 等等
?： 匹配0个或1个字符	如a?c 表示a与c之间可以有一个长度的任意字符, 如abc, adc, aec, a_c等等, 当a与c之间没有其他字符也是合法的。
[]： 匹配方括号中任意一个字符	如a[xyz]c 表示a与c之间必须也只能有一个字符, 但只能是 x 或 y 或 z, 如: axc, ayc, azc。
{ }： 取大括号内的集合	如： {string1,string2,...} 表示匹配 string1 或 string2 (或更多)其一字符串 , a{abc,xyz,123}b 表示a与b之间只能是abc或xyz或123这三个字符串之一。

Shell脚本

shell脚本



```
user@os-virtual-machine: ~/Desktop
user@os-virtual-machine:~/Desktop$
```

效率太低

将代码放入**脚本文件**中，这样每次只要执行脚本即可。

shell脚本的创建

新建一个文本文件，并命名为“test.sh”。以“Hello World”为例：

```
1  #!/bin/bash
2  echo "Hello World!"
```

- 1、“#!”是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，也就是说，使用哪一种Shell；
- 2、/bin/bash则是指明了解释器的具体位置。

shell脚本的运行

1、在新进程中运行：

(1) 将shell脚本作为程序运行，需要使用chmod命令给Shell脚本加上执行权限

```
user@os-virtual-machine:~/Desktop$ chmod +x ./test.sh
user@os-virtual-machine:~/Desktop$ ./test.sh
"Hello World!"
```

其中“chmod +x”表示给test.sh增加执行权限，“./”表示当前目录，整条命令的意思是执行当前目录下的test.sh脚本。

(2) 将Shell脚本作为参数传递给Bash解释器

```
user@os-virtual-machine:~/Desktop$ bash test.sh
"Hello World!"
```

通过这种方式运行脚本时，不需要在脚本文件的第一行指定解释器信息。

shell脚本的运行

2、在当前进程中运行：source命令，会读取脚本文件中的代码，并依次执行所有语句。也可以理解为source命令会强制执行脚本文件中的全部命令，而忽略脚本文件的权限：

source filename

或

. filename

```
user@os-virtual-machine:~/Desktop$ source ./test.sh
"Hello World!"
user@os-virtual-machine:~/Desktop$ source test.sh
"Hello World!"
user@os-virtual-machine:~/Desktop$ . ./test.sh
"Hello World!"
user@os-virtual-machine:~/Desktop$ . test.sh
"Hello World!"
```

使用source命令不用给脚本增加执行权限，并且写不写./都行。

shell脚本的调试：打印刚刚发生了什么

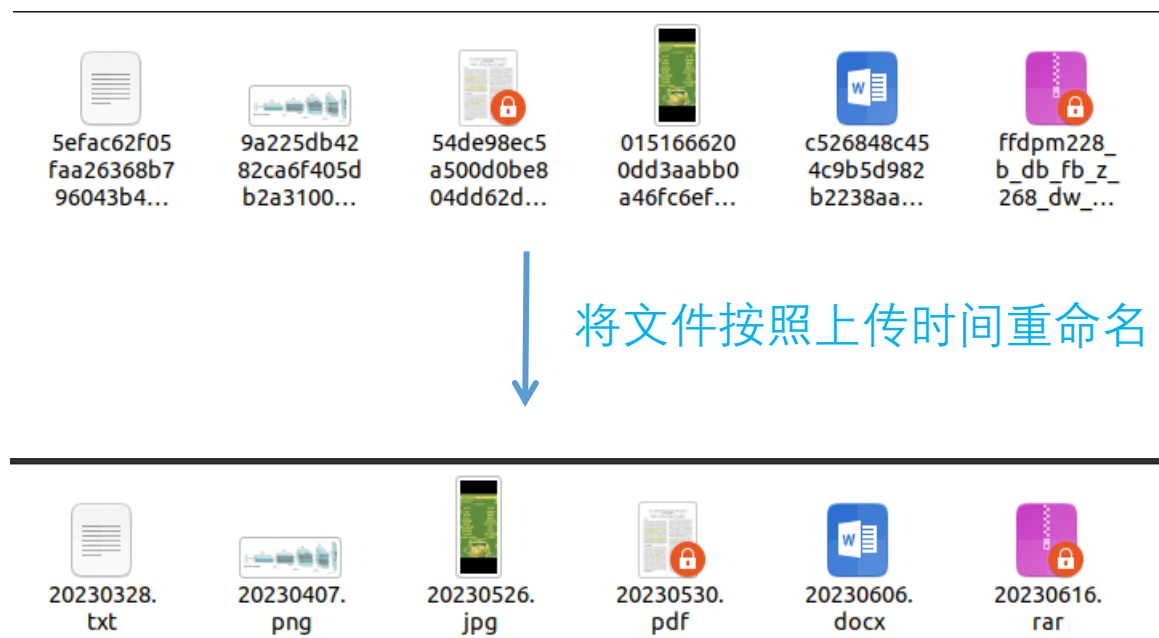
1、在运行脚本程序时加入调试选项： `bash -选项 ./xxxx.sh`

-e	如果一个命令返回一个非0退出状态值(失败),就退出。
-n	不需要执行脚本只是检查语法结构，返回所有的语法错误信息。
-u	置换时把未设置的变量看作出错。
-v	当读入shell输入行时，把它们显示出来。
-x	执行命令时，把命令和它们的参数显示出来。

2、在脚本程序中通过set命令调试程序：

`set - 选项， set +选项` #set命令的常用选项同上

Shell实战：批量文件重命名



如何用shell完成批量文件重命名呢？

- ① 循环和通配符
- ② 文件的修改时间提取
- ③ 字符串的拼接

其中一种实现方式：

```
1  #!/bin/bash
2
3  for i in `ls`
4  do
5      #提取文件名
6      filename=$i
7      #获取文件的修改时间
8      mtime=`stat -c %Y $filename`
9      #格式化时间
10     formate_date=`date '+%Y%m%d' -d @$mtime`
11     #提取文件的后缀名
12     name=${filename#*.}
13     #进行拼接
14     mv $i ${formate_date}.${name}
15 done
```




Shell扩展阅读

1、**shell的命令**：即ls/cd等linux命令，详细可参考

<http://codetoolchains.readthedocs.io/en/latest/4-Linux/2-shellcmd/index.html>

2、**shell解释器**：即sh/bash/csh等shell应用程序，详细可参考

<http://codetoolchains.readthedocs.io/en/latest/4-Linux/1-shellenv/1-shellsoft/index.html>

3、**shell语法**：即数据类型/变量/控制流语句/函数等编程语法，详细可参考

https://shellscript.readthedocs.io/zh_CN/latest/



本章作业

观察并完成实验报告

1. 找到5个可以持续运行的命令，观察他们进程状态的变化
2. 编写一个应用程序，观察应用程序中函数调用对进程状态的影响

程序设计

利用脚本编程实现对一个文件夹下的所有文件批量重命名。命名后的名字是文件的创建时间，其扩展名保持不变



感谢阅读
