



第9讲 Shell 编程

主讲：王洪泊





部分优秀作业

>> 优秀作品

作业标题

第一讲课件相关视频:××××-观后感

全选 <input type="checkbox"/>	真实姓名	提交时间	分数	批阅人	查看
<input type="checkbox"/>	唐梦涛	2021年3月21日 15:02:51	5.0	王洪泊	→
<input type="checkbox"/>	赵润哲	2021年3月30日 22:02:12	5.0	王洪泊	→
<input type="checkbox"/>	成佳鑫	2021年3月26日 9:58:40	5.0	王洪泊	→
<input type="checkbox"/>	任锦	2021年3月30日 19:43:00	5.0	王洪泊	→
<input type="checkbox"/>	孙清鑫	2021年3月22日 9:59:31	5.0	王洪泊	→
<input type="checkbox"/>	梁晨	2021年3月21日 20:28:31	5.0	王洪泊	→
<input type="checkbox"/>	王昊	2021年3月30日 21:51:09	5.0	王洪泊	→
<input type="checkbox"/>	丁鑫龙	2021年3月25日 14:14:45	5.0	王洪泊	→
<input type="checkbox"/>	章子鑫	2021年3月30日 23:18:48	5.0	王洪泊	→
<input type="checkbox"/>	何思宇	2021年3月20日 13:47:23	5.0	王洪泊	→
<input type="checkbox"/>	吴雨泽	2021年3月20日 21:48:02	5.0	王洪泊	→
<input type="checkbox"/>	徐子傲	2021年3月20日 5:23:28	5.0	王洪泊	→
<input type="checkbox"/>	吴海扬	2021年3月23日 16:00:26	5.0	王洪泊	→
<input type="checkbox"/>	裴俊杰	2021年3月20日 16:58:51	5.0	王洪泊	→

共14条记录 [首页] [尾页] 跳至第1页





>> 优秀作品

作业标题	Linux 实验三（综合）用Linux命令行操作处理日常业务				
全选 <input type="checkbox"/>	真实姓名	提交时间	分数	批阅人	查看
<input type="checkbox"/>	封华	2020年4月1日 23:08:52	5.0	王洪泊	➔
<input type="checkbox"/>	黄子悦	2020年3月29日 20:42:06	5.0	王洪泊	➔
<input type="checkbox"/>	王佳敏	2020年3月30日 19:13:13	5.0	王洪泊	➔
<input type="checkbox"/>	陈洁	2020年4月1日 10:20:30	5.0	王洪泊	➔
<input type="checkbox"/>	张云茹	2020年4月1日 23:20:46	5.0	王洪泊	➔
<input type="checkbox"/>	金玉卿	2020年4月5日 22:43:31	5.0	王洪泊	➔
<input type="checkbox"/>	张梦昊	2020年4月2日 17:46:29	5.0	王洪泊	➔
<input type="checkbox"/>	陈璐	2020年4月3日 11:57:41	5.0	王洪泊	➔

共8条记录 [首页] [尾页] 跳至第 1 ▼ 页

一枝独秀固然美，万紫千红方为春。





» 优秀作品

作业标题	Linux 实验四 源代码安装Apache服务器				
全选 <input type="checkbox"/>	真实姓名	提交时间	分数	批阅人	查看
<input type="checkbox"/>	米冠飞	2020年3月31日 14:22:55	5.0	王洪泊	➔
<input type="checkbox"/>	葛启丰	2020年3月25日 16:13:17	5.0	王洪泊	➔
<input type="checkbox"/>	陈肇宁	2020年3月26日 14:21:05	5.0	王洪泊	➔
<input type="checkbox"/>	何昕宇	2020年3月31日 19:18:28	5.0	王洪泊	➔
<input type="checkbox"/>	温玥明	2020年4月2日 16:40:49	5.0	王洪泊	➔
<input type="checkbox"/>	张云茹	2020年4月3日 23:38:19	5.0	王洪泊	➔
<input type="checkbox"/>	黄子悦	2020年4月2日 13:55:05	5.0	王洪泊	➔

共7条记录 [首页] [尾页] 跳至第 页

送人玫瑰，手有余香！





《Linux操作系统》结课考试将于下周四(4.22)上午10:00-11:30举行，其中考场待定。

- ☐ A 在学校，按时到指定教室，参加考场
- ☐ B 防疫未返校，可以按时通过网上参加考试

提交





本章学习要点

- (1) Shell编程基本步骤
- (2) Shell变量
- (3) 表达式与运算符
- (4) 流程控制语句
- (5) 函数

建议课时：2课时





9.1 Shell编程基本步骤

9.1.1 编写Shell脚本

● 脚本的编写

- Shell脚本本身就是一个文本文件，示例：
#!/bin/bash
#显示 "Hello World!"
echo "Hello World!"
- 与其他脚本语言编程一样，Shell脚本编程无需编译器，也不需要集成开发环境，一般使用文本编辑器即可。
- 多数Shell程序员首选的编辑器是Vi或Emacs，在桌面环境中可直接使用图形化编辑器gedit或kate。





9.1 Shell编程基本步骤

9.1.1 编写Shell脚本

● Shell脚本的基本构成

■ Shell脚本构成示例:

```
#!/bin/bash
#这是一个测试脚本
echo -n "当前日期和时间: "
date
echo -n "程序执行路径: "$PATH
echo "当前登录用户名: `whoami`"
echo -n "当前目录:"
pwd
#end
```





9.1 Shell编程基本步骤

9.1.1 编写Shell脚本

● Shell脚本的基本构成

- 通常在第1行以“#!”开头指定Shell脚本的运行环境，即声明该脚本使用哪个Shell程序运行。如果没有指定，则以当前正在执行的Shell来解释执行。
- 以“#”开头的行是注释行。
- 与其他编程语言一样，Shell会忽略空行。
- echo命令用来显示提示信息。
- “whoami”字符串左右的反引号（`）用于命令转换（转换）。





9.1 Shell编程基本步骤

9.1.1 编写Shell脚本

● 包含外部脚本

- 包含外部脚本文件的用法如下：
 - . 脚本文件名
 - 或
 - source** 脚本文件名
- 两种方式的作用一样，为简单起见，一般使用点号，但要注意点号和脚本文件名之间一定要有一个空格。





9.1 Shell编程基本步骤

9.1.2 执行Shell脚本

● 1. 在命令行提示符下直接执行

- 将Shell脚本文件的权限设置为可执行，然后在命令行提示符下直接执行它。例如：

```
chmod +x example1
```

- 直接编辑生成的脚本文件并没有执行权限。
- 如果包含由外部脚本，被包含的脚本并不需要有执行权限。
- 直接输入脚本文件名，需要让该脚本所在的目录被包含在环境变量PATH所定义的命令搜索路径（PATH）中，否则要明确指定脚本文件的路径。





9.1 Shell编程基本步骤

9.1.2 执行Shell脚本

● 2. 在指定的Shell下执行脚本

- 在指定的Shell下执行脚本，以脚本名作为参数：
Shell名称 脚本名 [参数]
- 这种运行方式是直接运行Shell解释器，其参数就是Shell脚本的文件名。

● 3. 将输入重定向到Shell脚本

- 让Shell从指定文件中读入命令行，并进行相应处理：
bash < 脚本名





9.1 Shell编程基本步骤

9.1.3 调试Shell脚本

- 在**bash**中**Shell**脚本的调试主要是利用**bash**命令解释程序的选项来实现。其格式为：
 bash [选项] 脚本名
- **-v**选项允许用户查看**Shell**程序的读入和执行，会打印出命令行的原始内容。
- **x**选项允许用户查看**Shell**程序的执行，会打印出经过替换后的命令行的内容。





9.2 Shell 变量

9.2.1 变量类型

- 用户自定义变量

- 它是在编写Shell脚本时定义的，可以在Shell程序内任意使用和修改。可以将它看作局部变量，仅在当前Shell实例中有效。

- 环境变量

- 作为系统环境的一部分，不必去定义它们，可以在Shell程序中使用它们，某些变量（如PATH）可以在Shell中加以修改。可以将它看作全局变量。

- 内部变量

- 此类变量在程序中用来做出判断。在Shell程序内，这类变量的值是不能修改的。





9.2 Shell 变量

9.2.2 变量赋值和访问

● 变量定义

- 变量无需事先声明，给变量赋值也就是定义一个变量的过程：
变量名=值
- 变量名的命名应当遵循如下规则：
 - 首个字符必须为字母（**a-z, A-Z**）
 - 中间不能有空格，可以使用下划线（**_**）
 - 不能使用标点符号
 - 不能使用Shell中的关键字





9.2 Shell 变量

9.2.2 变量赋值和访问

● 变量引用

- 如果要访问变量值，可以在变量名前面加一个美元符号 “\$”。
- 通常使用函数 **echo** 来显示变量。
- 变量名加花括号 “{}” 是可选的。

● 只读变量

- 使用 **readonly** 命令可以将变量定义为只读变量，只读变量的值不能被改变。

● 删除变量

- 使用 **unset** 命令可以删除变量。

● 添加环境变量

- 使用 **export** 命令将变量添加到环境中，作为临时的环境变量（一种全局变量）。





9.2 Shell 变量

9.2.3 内部变量

变量	说 明
\$0	当前脚本的文件名
\$n	传递给脚本或函数的参数。n是一个数字，表示第几个参数。例如，第一个参数是\$1，第二个参数是\$2，以此类推
\$#	传递给脚本或函数的参数个数
\$*	传递给脚本或函数的所有参数
\$@	传递给脚本或函数的所有参数。被双引号（" "）包含时，与 \$* 稍有不同
\$?	上个命令的退出状态，或函数的返回值
\$\$	当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID





9.2 Shell 变量

9.2.4 位置参数

- 当编写一个带有若干参数的Shell脚本时，可以用命令行或从其他的Shell脚本调用位置参数。
- 位置参数使用系统给出的专用名，存放在变量中的第1个参数名为1，可以用\$1来访问；第2个参数名为2，可以利用\$2来访问它，依此类推。当参数超过10个时，要用花括号将参数序号括起来，如\${12}。
- \$0是一个比较特殊的位置参数，用于表示脚本自己的文件名。
- \$*和\$@ 都表示传递给函数或脚本的所有参数。
- \$#是指传递参数的个数。
- 调用Shell程序可以省略位置居后的位置参数。
- 在Shell程序中，可以利用set命令为位置参数赋值或重新赋值：

set [参数列表]





9.2 Shell 变量

9.2.5 变量值输出

● echo命令

- Shell变量可以使用echo命令实现标准输出，在屏幕上打印出指定的字符串。
- echo可以用来实现更复杂的输出格式控制。
- 输出内容使用双引号将阻止Shell对大多数特殊字符进行解释，但美元符号（\$）、反引号（`）和双引号（"）仍然保持其特殊意义，如果要在双引号中的内容中显示这些符号，需要使用转义符。

● printf命令

- printf命令用于格式化输出。
- printf命令可以提供格式控制字符串，语法如下：
printf 格式字符串 [参数列表...]





9.2 Shell 变量

9.2.6 变量值读取

- 使用**read**命令可以将变量的值作为字符串从键盘读入：
 read 变量
- 在执行**read**命令时可以不指定变量参数，它会将接收到的数据放置在环境变量**\$REPLY** 中。
- **read**读入的变量可以有多个，第一个数据给第一个变量，第二个数据给第二个变量，如果输入数据个数过多，则最后所有的值都给第一个变量。下面的脚本示例读取两个数，并显示出来。
- 还可以使用选项**-n**对输入的字符进行计数，当输入的字符数目达到预定数目时，自动退出，并将输入的数据赋值给变量。





9.2 Shell 变量

9.2.7 变量替换

- 可以使用以下几种变量替换形式。

`${var}`: 替换为变量本来的值。

`${var:-word}`: 如果变量`var`为空或已被删除, 则返回`word`, 但不改变`var`的值。

`${var:=word}`: 如果变量`var`为空或已被删除, 则返回`word`, 并将`var`的值设置为 `word`。

`${var:?message}`: 如果变量`var` 为空或已被删除, 则将消息`message`发送到标准错误输出, 可以用来检测变量`var`是否可以被正常赋值。这种替换出现在Shell脚本中, 脚本将停止运行。

`${var:+word}`: 如果变量`var` 被定义, 则返回 `word`, 但不改变 `var` 的值。





9.2 Shell 变量

9.2.8 数组

- **bash**支持一维数组（不支持多维数组。
- 在**Shell**中用括号来表示数组，数组元素用空格符号分割开：
数组名=(值1 ... 值n)
- 也可以单独定义数组的各个元素。
- 可以不使用连续的下标，而且下标的范围没有限制。
- 读取数组元素值的语法格式为：
\${数组名[下标]}
- 采用以下用法获取数组元素的个数：
\${#数组名[@]}
- 以下方法用于取得数组单个元素的长度：
\${#数组名[n]}





9.3 表达式与运算符

9.3.1 表达式

● 算术表达式

- bash自身并不支持简单的数学运算，但是可以通过awk 和expr等命令来实现数学运算。
- expr最为常用，使用它能够完成表达式的求值操作。例如：
`expr 5 + 3`
- 注意操作数（用于计算的数）与运算符之间一定要有空格。
- 更为简单的方式是使用`$[]`表达式进行数学计算，例如
`val=$((5+3))`
- 这种形式不要求运算符与操作数之间有空格。





9.3 表达式与运算符

9.3.1 表达式

● 逻辑表达式

- 逻辑表达式主要用于条件判断，值为**true**（或0）表示结果为真；值为**false**（非零值）表示结果为假。
- 通常使用**test**命令来判断表达式的真假。语法格式如下
test 逻辑表达式
- Linux每个版本中都包含**test**命令，但该命令有一个更常用的别名，即左方括号 “[”。语法格式如下
[逻辑表达式]
- 当使用左方括号而非**test**时，其后必须始终跟着一个空格、要评估的逻辑表达式、一个空格和右方括号，右方括号表示所需评估表达式的结束。逻辑表达式两边的空格是必需的。





9.3 表达式与运算符

9.3.2 算术运算符

- $+$: 加法
- $-$: 减法
- $*$: 乘法
- $/$: 除法
- $\%$: 取余
- $=$: 赋值





请你猜一猜

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example0.sh  
hong@LAPTOP-DTA58180:~$ cat ./example0.sh  
x=8  
y=$x  
z=$x+$y  
m=$x-$y  
n=$x\*$y  
echo $y  
echo $z  
echo $m  
echo $n  
  
hong@LAPTOP-DTA58180:~$
```

❖ 右图四句输出
分别是什么？
(请弹幕)

❖ ?

❖ ?

❖ ?

❖ ?





你猜对了吗？

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180: ~$ vim example0.sh  
hong@LAPTOP-DTA58180: ~$ cat ./example0.sh  
x=8  
y=$x  
z=$x+$y  
m=$x-$y  
n=$x\*$y  
echo $y  
echo $z  
echo $m  
echo $n  
  
hong@LAPTOP-DTA58180: ~$ bash example0.sh  
8  
8+8  
8-8  
8*8  
hong@LAPTOP-DTA58180: ~$
```

❖ 右图四句输出
分别是什么？
(请弹幕)

❖ 8

❖ 8+8

❖ 8-8

❖ 8*8



如何得到我们想要的结果？（1）

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example1.sh  
hong@LAPTOP-DTA58180:~$ cat ./example1.sh  
x=8  
y=$x  
z=$(( $x + $y ))  
m=$(( $x - $y ))  
n=`expr $x \* $y`  
echo $y  
echo $z  
echo $m  
echo $n  
  
hong@LAPTOP-DTA58180:~$ bash example1.sh  
8  
16  
0  
64  
hong@LAPTOP-DTA58180:~$
```



如何得到我们想要的结果？（2）

```
hong@LAPTOP-DTA5818O: ~  
hong@LAPTOP-DTA5818O:~$ vim example2.sh  
hong@LAPTOP-DTA5818O:~$ cat example2.sh  
x=8  
y=$x  
z=${x + $y}  
let m=x-$y  
n=`expr $x \* $y`  
echo $y  
echo $z  
echo $m  
echo $n  
  
hong@LAPTOP-DTA5818O:~$ bash example2.sh  
8  
16  
0  
64  
hong@LAPTOP-DTA5818O:~$
```





9.3 表达式与运算符

9.3.3 整数关系运算符

- `-eq`: 相等。检测两个数是否相等，相等返回 `true`。
- `-ne`: 不等于。检测两个数是否相等，不相等返回 `true`。
- `-gt`: 大于。检测左边的数是否大于右边的，如果是，则返回 `true`。
- `-lt`: 小于。检测左边的数是否小于右边的，如果是，则返回 `true`。
- `-ge`: 大于等于。
- `-le`: 小于等于。





关系运算符示例

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example3.sh  
hong@LAPTOP-DTA58180:~$ cat example3.sh  
#!/bin/bash  
a=10  
b=20  
if [ $a -eq $b ]  
then  
    echo "$a -eq $b : a 等于 b"  
else  
    echo "$a -eq $b : a 不等于 b"  
fi  
hong@LAPTOP-DTA58180:~$ bash example3.sh  
10 -eq 20 : a 不等于 b  
hong@LAPTOP-DTA58180:~$
```





9.3 表达式与运算符

9.3.4 字符串检测运算符

- **=**: 检测两个字符串是否相等, 相等返回**true**。
- **!=**: 检测两个字符串是否相等, 不相等返回**true**。
- **-z**: 检测字符串长度是否为0, 为0返回 **true**。
- **-n**: 检测字符串长度是否为0, 不为0返回 **true**。
- **str**: 检测字符串是否为空, 不为空返回 **true**。这里不用使用运算符, 直接测试字符串, 如 **[\$a]** 返回 **true**。



字符串检测运算符示例

```
选择hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ cat example4.sh  
#!/bin/bash  
a="abc"  
b="xyz"  
if [ $a = $b ]  
then  
    echo "$a = $b : a等于 b"  
else  
    echo "$a= $b : a不等于b"  
fi  
if [ -z $a ]  
then  
    echo "-z $a :字符串长度为0 "  
else  
    echo "-z $a : 字符串长度不为0"  
fi  
if [ $a ]  
then  
    echo "$a : 字符串非空"  
else  
    echo "$a : 字符串为空"  
fi  
hong@LAPTOP-DTA58180:~$ bash example4.sh  
abc= xyz : a不等于b  
-z abc : 字符串长度不为0  
abc : 字符串非空  
hong@LAPTOP-DTA58180:~$
```





9.3 表达式与运算符

9.3.5 文件测试运算符

- **-b**: 检测文件是不是块设备文件，如果是，则返回 **true**。
- **-c**: 检测文件是不是字符设备文件，如果是，则返回 **true**。
- **-d**: 检测文件是不是目录，如果是，则返回 **true**。
- **-f**: 检测文件是不是普通文件（既非目录，又非设备文件），如果是，则返回 **true**。
- **-g**: 检测文件是否设置了 **SGID** 位，如果是，则返回 **true**。
- **-k**: 检测文件是否设置了粘着位（**Sticky Bit**），如果是，则返回 **true**。
- **-p**: 检测文件是不是具名管道，如果是，则返回 **true**。
- **-u**: 检测文件是否设置了 **SUID** 位，如果是，则返回 **true**。
- **-r**: 检测文件是否可读，如果是，则返回 **true**。
- **-w**: 检测文件是否可写，如果是，则返回 **true**。
- **-x**: 检测文件是否可执行，如果是，则返回 **true**。
- **-s**: 检测文件是否为空（文件大小是否大于0），不为空返回 **true**。
- **-e**: 检测文件（包括目录）是否存在，如果是，则返回 **true**。



检测文件的读写属性示例

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example5.sh  
hong@LAPTOP-DTA58180:~$ cat example5.sh  
#!/bin/bash  
file="./example4.sh"  
if [ -r $file ]  
then  
    echo "文件具有读取权限"  
else  
    echo "文件并不具有读取权限"  
fi  
  
hong@LAPTOP-DTA58180:~$ bash example5.sh  
文件具有读取权限  
hong@LAPTOP-DTA58180:~$ ls -l  
total 0  
-rw-rw-rw- 1 hong hong 68 Apr  5 10:46 example0.sh  
-rw-rw-rw- 1 hong hong 86 Apr  5 11:11 example1.sh  
-rw-rw-rw- 1 hong hong 85 Apr  5 11:21 example2.sh  
-rw-rw-rw- 1 hong hong 119 Apr  5 15:22 example3.sh  
-rw-rw-rw- 1 hong hong 300 Apr  5 15:49 example4.sh  
-rw-rw-rw- 1 hong hong 136 Apr  5 16:04 example5.sh  
hong@LAPTOP-DTA58180:~$
```





9.3 表达式与运算符

9.3.6 布尔运算符

- **-a**: 与运算。两个表达式都为 **true**才返回**true**。
- **-o**: 或运算。有一个表达式为**true**就返回 **true**。
- **!**: 非运算。表达式值为 **true**则返回**false**， 否则返回 **true**。

```
hong@LAPTOP-DTA58180: ~  
OP-DTA58180:~$ vim example6.sh  
hong@LAPTOP-DTA58180:~$ cat example6.sh  
#!/bin/bash  
a=5  
b=10  
if [ $a -lt 10 -a $b -gt 15 ]  
then  
    echo "两个条件都满足"  
else  
    echo "不是两个条件都满足"  
fi  
hong@LAPTOP-DTA58180:~$ bash example6.sh  
不是两个条件都满足  
hong@LAPTOP-DTA58180:~$
```





9.4 流程控制语句

9.4.1 条件语句

● if语句

- if语句通过判定条件表达式作出选择。
- 大多数情况下，可以使用**test**命令来对条件进行测试。
- 实际应用中通常用方括号来代替**test**命令。

(1) if ... else 语句

if [条件表达式]

then

语句序列

fi





9.4 流程控制语句

9.4.1 条件语句

- **if语句**

(2) if ... else ... fi 语句

if [条件表达式]

then

语句序列1

else

语句序列2

fi





9.4 流程控制语句

9.4.1 条件语句

● if语句

(3) if ... elif ... fi 语句

```
if [ 条件表达式1 ]  
then  
    语句序列1  
elif [ 条件表达式2 ]  
then  
    语句序列2  
elif [ 条件表达式3 ]  
then  
    语句序列3  
.....  
else  
    语句序列n  
fi
```



9.4 流程控制语句

9.4.1 条件语句

● case语句

case 值 in

模式1)

语句序列1

; ;

模式1)

语句序列2

; ;

.....

模式n)

语句序列n

; ;

*)

其他语句序列

esac

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example7.sh  
hong@LAPTOP-DTA58180:~$ cat example7.sh  
#!/bin/bash  
case $USER in  
hong)  
    echo "Welcome Teacher hong!"  
    ;;  
zhang|li)  
    echo "Welcome student!"  
    ;;  
root)  
    echo "Welcome Super administrator!"  
    ;;  
*)  
    echo "欢迎 $USER !";;  
esac  
hong@LAPTOP-DTA58180:~$ bash example7.sh  
Welcome Teacher hong!  
hong@LAPTOP-DTA58180:~$ sudo bash example7.sh  
Welcome Super administrator!  
hong@LAPTOP-DTA58180:~$
```



9.4 流程控制语句

9.4.2 循环结构

● 循环语句while

- while循环用于不断执行一系列命令，直到测试条件为假（false）。
while 测试条件
do
 语句序列
done

```
选择hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180: ~$ vim example8.sh  
hong@LAPTOP-DTA58180: ~$ cat example8.sh  
#!/bin/bash  
total=0  
num=0  
while [ $num -le 100 ]  
do  
    total=`expr $total + $num`  
    num=`expr $num + 1`  
done  
echo "结果等于: $total"  
hong@LAPTOP-DTA58180: ~$ bash example8.sh  
结果等于: 5050  
hong@LAPTOP-DTA58180: ~$
```



9.4 流程控制语句

9.4.2 循环结构

● 循环语句 **until**

- **until** 循环用来执行一系列命令，直到所指定条件为真时才终止循环。

```
until 测试条件
do
    语句序列
done
```

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example9.sh  
hong@LAPTOP-DTA58180:~$ cat example9.sh  
#!/bin/bash  
total=0  
num=0  
until [ $num -gt 100 ]  
do  
    total=`expr $total + $num`  
    num=`expr $num + 1`  
done  
echo "计算结果等于: $total"  
hong@LAPTOP-DTA58180:~$ bash example9.sh  
计算结果等于: 5050  
hong@LAPTOP-DTA58180:~$
```



9.4 流程控制语句

9.4.2 循环结构

● 循环语句for

- for循环适用于明确知道重复执行次数的情况，它将循环次数通过变量预先定义好，实现使用计数方式控制循环。

for 变量 [in 列表]

do

语句序列

done

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example10.sh  
hong@LAPTOP-DTA58180:~$ cat example10.sh  
#!/bin/bash  
for FILE in $HOME/*.sh  
do  
    echo $FILE  
done  
hong@LAPTOP-DTA58180:~$ bash example10.sh  
/home/hong/example0.sh  
/home/hong/example10.sh  
/home/hong/example1.sh  
/home/hong/example2.sh  
/home/hong/example3.sh  
/home/hong/example4.sh  
/home/hong/example5.sh  
/home/hong/example6.sh  
/home/hong/example7.sh  
/home/hong/example8.sh  
/home/hong/example9.sh  
hong@LAPTOP-DTA58180:~$
```





9.4 流程控制语句

9.4.2 循环结构

● 其他循环语句

- **break**语句用来终止一个重复执行的循环。这种循环可以是**for**、**until**或者**while**语句构成的循环。

`break [n]`

- **continue**语句跳过循环体中位于它后面的语句，回到本层循环的开头，进行下一次循环。

`continue [n]`

- **exit**语句用来退出一个**Shell**程序，并设置退出值。

`exit [n]`





9.5 函数

9.5.1 函数的定义和调用

● 函数定义

- Shell 函数必须先定义后使用。

```
[function] 函数名()  
{  
    命令序列  
    [return 返回值]  
}
```

● 函数调用

- 调用函数只需要给出函数名，不需要加括号，就像一般命令那样使用。

函数名 参数1 参数2 参数n





9.5 函数

9.5.2 函数的返回值

● 1.使用全局变量

- 先定义一个变量，用来接收函数的计算结果，脚本在需要的时候访问这个变量来获得函数的返回值。
- 使用变量要注意不要修改父脚本里不期望被修改的内容。

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example11.sh  
hong@LAPTOP-DTA58180:~$ cat example11.sh  
#!/bin/bash  
Hello() {  
mystr='Hello World!'  
}  
Hello  
echo $mystr  
hong@LAPTOP-DTA58180:~$ bash example11.sh  
Hello World!  
hong@LAPTOP-DTA58180:~$
```



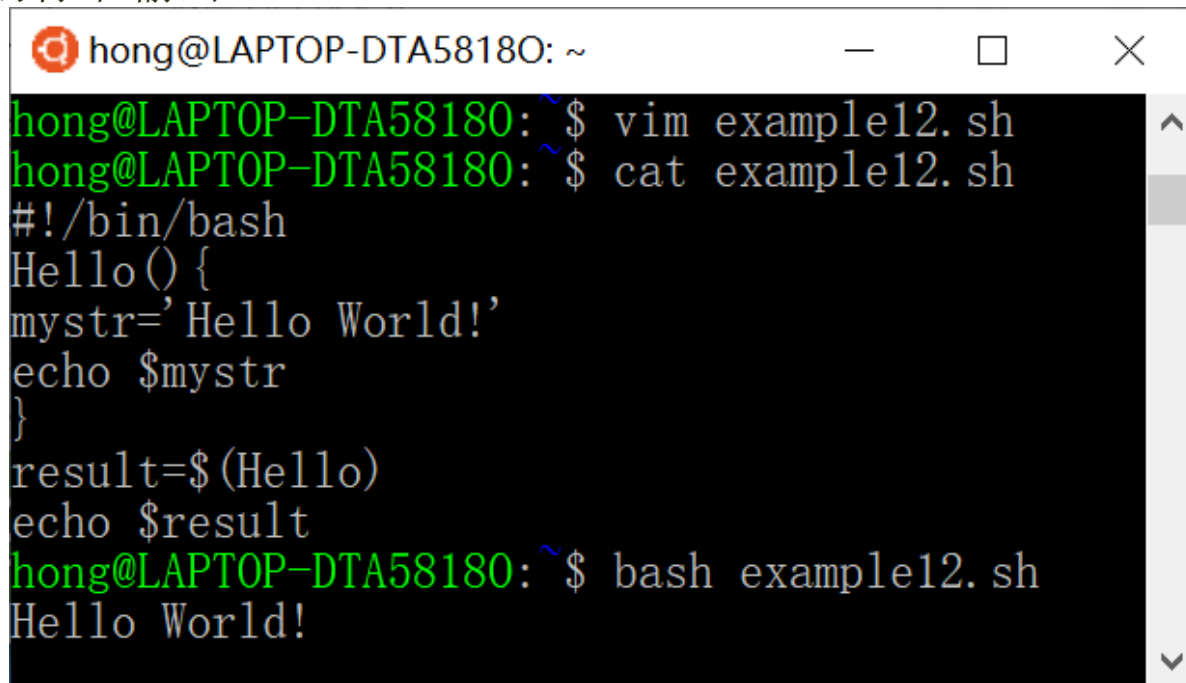


9.5 函数

9.5.2 函数的返回值

●2. 在函数中使用标准输出

- 将一个Shell函数作为一个子程序调用（命令替换），将返回值写到子程序的标准输出。



```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example12.sh  
hong@LAPTOP-DTA58180:~$ cat example12.sh  
#!/bin/bash  
Hello() {  
  mystr='Hello World!'  
  echo $mystr  
}  
result=$(Hello)  
echo $result  
hong@LAPTOP-DTA58180:~$ bash example12.sh  
Hello World!
```





9.5 函数

9.5.2 函数的返回值

●3. 在函数中使用**return**返回整数值

- `$?`是一个特殊的内部变量，可用于获取上一个命令执行后的返回结果，可以直接通过函数**return**语句来接收返回值。

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ vim example13.sh  
hong@LAPTOP-DTA58180:~$ cat example13.sh  
#!/bin/bash  
addnum() {  
    val=`expr $1 + $2`  
    return $val  
}  
addnum 4 5  
ret=$?  
echo "两个数的和是: $ret !"  
hong@LAPTOP-DTA58180:~$ bash example13.sh  
两个数的和是: 9 !  
hong@LAPTOP-DTA58180:~$
```




9.5 函数

9.5.3 函数参数

- 在Shell中调用函数时可以向其传递参数。
- 与脚本一样，在函数体内部也是通过\$*n*的形式来获取参数的值。
- 例如，\$1表示第一个参数，\$2表示第二个参数。







9.6.1 shell的启动与退出过程

- shell的启动流和工作过程如上图所示。用户登录过程中，在输入用户名和密码后，系统要检查以下文件：
 - /etc/passwd: 系统用户数据库。
 - /etc/shadow: 影子密码和用户登录控制文件。
 - /etc/group: 系统组定义文件。
- 如果用户名和密码合法，则启动shell，否则要求用户重新输入用户名和密码。





shell的启动过程（续）

- ❖ 在系统后，用户登录系统时，**shell**启动过程中，依次进行以下操作：
 - (1) 执行系统脚本文件/etc/profile;
 - (2) 在它的控制下再执行/etc/profile.d/目录的*.sh脚本文件;
 - (3) 执行用户profile文件~/.bash_profile;
 - (4) 如果~/.bashrc存在则执行之;
 - (5) 如果/etc/bashrc文件存在则执行之。
- ❖ 需要说明的是，以上步骤都是在当前**shell**内执行的（参见后续章节），目的是为用户设置环境变量或做相关准备。一切执行完毕之后出现系统提示符。





shell的退出

- ❖ 当用户通过（Ctrl_D、exit或logout）注销时，将调用用户家目录内的脚本文件.bash_logout
- ❖ 若用户想让shell退出时做些事情，比如删除自己创建的临时文件，则可将shell命令添加到.bash_logout文件。





9.6.2 修改 .bashrc 和 .bash_profile 文件

- ❖ 用户可在shell的启动过程加入自己的内容，比如设置自己所需的环境变量。用户也可以在.bashrc和.bash_profile中加入自己的用户内容，一般是放在文件的最后，使其在一切都准备之后执行之。
- ❖ 这样可以使用户在登录之后就进入某个规定程序，当其退出时用户也随之注销。因此可让用户只在某个规定封闭环境下工作，而不让其真正进入系统，从而也看不到系统的提示符，因而提高了系统的安全性。





修改 .bash_logout 文件

- ❖ 用户也做自己退出时做点工作。这些退出时做的工作是通过 .bash_logout 实现的。
- ❖ 也就是说如果用户想在退出时让系统做点工作的话，可以将做工作“命令”加入脚本文件 .bash_logout 中。



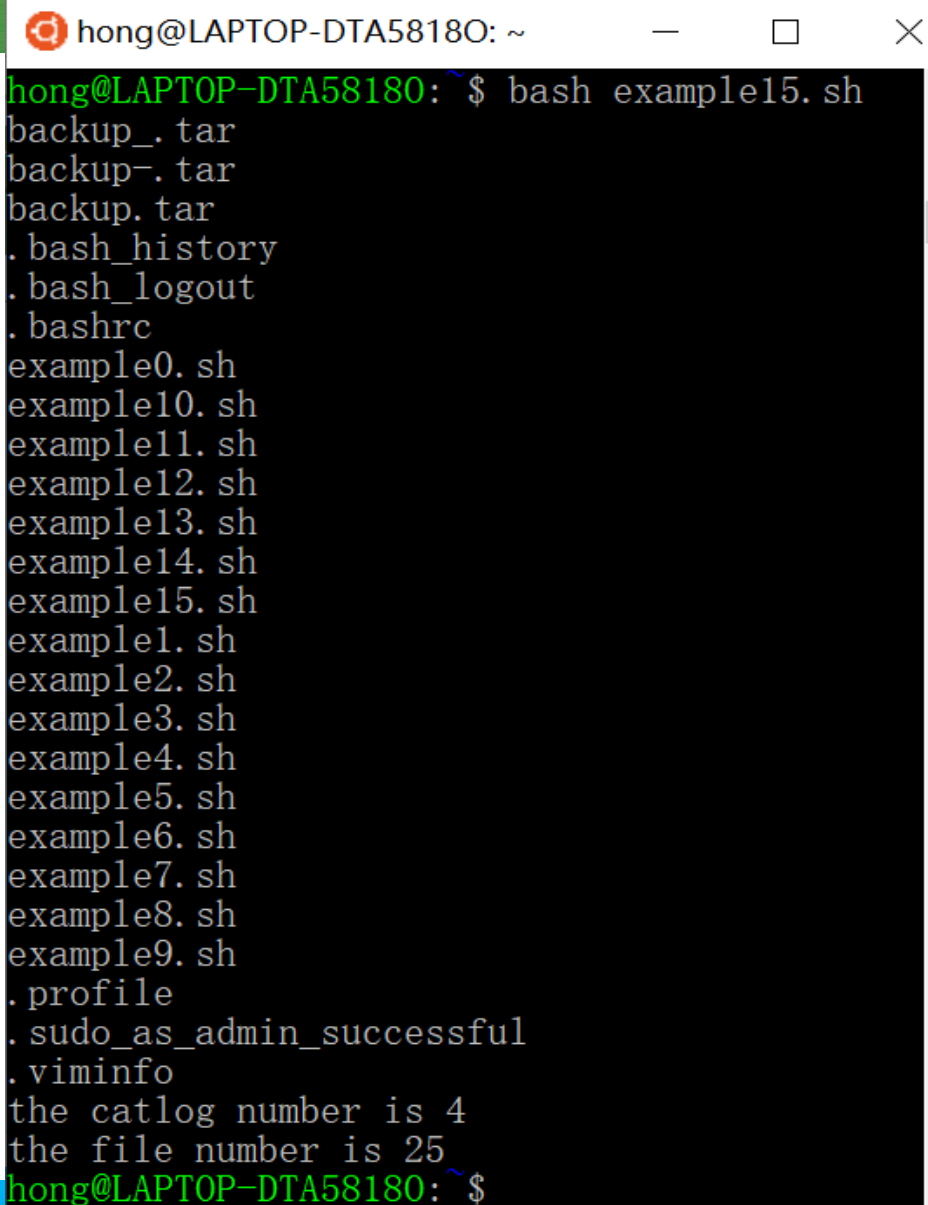


linux下Shell编程：输出当前目录下所有文件，并输出文件总数和目录总数

```
hong@LAPTOP-DTA58180: ~  
hong@LAPTOP-DTA58180:~$ cat example15.sh  
#!/bin/bash  
cd $1  
file=`ls -a`  
catlog=0  
filenum=0  
for loop in $file  
do  
if [ -d $loop ]  
then  
catlog=`expr $catlog + 1`  
else  
filenum=`expr $filenum + 1`  
echo "$loop"  
fi  
done  
echo -e "the catlog number is $catlog"  
echo -e "the file number is $filenum"  
hong@LAPTOP-DTA58180:~$
```



linux下Shell编程：输出当前目录下所有文件，并输出文件总数和目录总数



A terminal window titled 'hong@LAPTOP-DTA58180: ~' with standard window controls. The terminal shows a green prompt 'hong@LAPTOP-DTA58180:~\$' followed by the command 'bash example15.sh'. The output lists 25 files and directories in the current directory, including backup files, shell scripts (example0.sh to example15.sh, example1.sh to example9.sh), and system files (.bash_history, .bash_logout, .bashrc, .profile, .sudo_as_admin_successful, .viminfo). The final two lines of output are 'the catlog number is 4' and 'the file number is 25', followed by a green prompt 'hong@LAPTOP-DTA58180:~\$'.

```
hong@LAPTOP-DTA58180:~$ bash example15.sh
backup_.tar
backup-.tar
backup.tar
.bash_history
.bash_logout
.bashrc
example0.sh
example10.sh
example11.sh
example12.sh
example13.sh
example14.sh
example15.sh
example1.sh
example2.sh
example3.sh
example4.sh
example5.sh
example6.sh
example7.sh
example8.sh
example9.sh
.profile
.sudo_as_admin_successful
.viminfo
the catlog number is 4
the file number is 25
hong@LAPTOP-DTA58180:~$
```



shell 种 类

- UNIX/Linux中的shell有多种类型，其中最常用的几种是Bourne shell（sh或bsh）、Bourne again shell（bash），C shell（csh）、tc shell（tcsh）、Korn shell（ksh）和Z shell（zsh）等。





实验六 Shell 编程实践

- ❖ 编写Shell程序，实现每天12:30将主目录下的所有目录和文件归档并压缩为mybackup.tar.gz，然后让该脚本开机运行（提示：编辑/etc/init.d/rc.local文件来实现）。
- ❖ 注意步骤的分解，试着耐心地逐步实现上述功能。

❖ 参考资料：Linux下定时备份文件

https://blog.csdn.net/q290994/article/details/79186821?depth_1-utm_source=distribute.pc_relevant.none-task-blog-OPENSEARCH-3&utm_source=distribute.pc_relevant.none-task-blog-OPENSEARCH-3





问题答疑

- 使用echo命令清空日志文件的内容，命令格式如下所示。
 - echo > 日志文件
- 1. 遇到权限不够的提示，为什么，如何解决？权限不够加sudo啊，可是你会发现sudo cat /dev/null > /var/log/wtmp 一样会提示权限不够，为什么呢？
 - 因为sudo只能让cat命令以sudo的权限执行，而对于>这个符号并没有sudo的权限，我们可以使用sudo sh -c "cat /dev/null > /var/log/wtmp " 让整个命令都具有sudo的权限执行。
- 2. 为什么cleanlogs.sh可以将log文件清除？因为/dev/null，里面是空的，重定向到 /var/log/wtmp 文件后，就清空了 wtmp 文件的内容。
- -----
- 原文链接: <https://blog.csdn.net/flysnownet/java/article/details/90770625>

