Shell 编程

1. 什么是 shell 脚本？

   shell 脚本是包含了一系列命令的文件。shell 读取这个文件，然后执行这些命令。如：

```
#! /bin/bash
# This is our first script
echo 'Hello, World'
```

   使用 vi 建立上述文件，保存为 hello_world.sh。

```
chmod 755 hello_world.sh
./hello_world.sh
```

   #开头的内容是注释

   #!被称为 shebang。用来告知操作系统后面的脚本应该使用的解释器的名字。每个 shell 脚本都应该用其作为第一行。

2. 脚本的理想位置

   建议在~下建立文件夹 bin，将所有的脚本放入该目录。如果我们编写了所有用户都可以用的脚本，建议放在/usr/local/bin。

   管理员使用的脚本都放在/usr/local/sbin 下。

   对于编译好的程序，建议放在/usr/local 下。

3. 为编写脚本而配置 vim

   打开~/.vimrc 文件，添加下列配置

   :syntax on　　#打开语法高亮

   :set tabstop=4　#设置 Tab 键为 4 个空格

   :set autoindent　#开启自动缩进特性，让 vim 对新一行和上一行保持相同缩进

4. 变量和常量的定义

   变量名=值

   注意：等号两侧不能包含空格；变量值如果包含空格，需要用""包含起来；变量是没有类型的

   常量对于 shell 来讲，与变量没有区别。只是建议常量名全部大写。

   可以在一行中给多个变量赋值，如 a=5 b="a string"

   当使用一个变量时，在前面加上$，如$a

   例子：

```
a=z
b="a string"
c="a string and $b"
d=$(ls -l)    #()表示命令的结果
e=$((5*7))    #算数运算使用(())
f="\t\ta string\n"  #转义字符需要使用""
```

5. here 文档

   使用 here 文档，双引号和单引号将失去它们在 shell 中的特殊含义。

```
cat << _EOF_
$foo
"foo"
'$foo'
\$foo
_EOF_
```

   EOF 表示 End Of File，即文件结尾。必须出现在单独的一行，且结尾没有空格。

   如果在 here 文档中，将<<改为<<-，则忽略开头的 Tab 字符。

6. shell 函数

语法：

```
function name(){
       commands
       return
}
```

例子：

```
function funct(){
       echox "Step 2"
       return
}
```

调用：直接使用函数名称进行调用

```
funct
```

7. 局部变量

定义在所有函数之外的变量被称作全局变量。

在函数内部，使用 local 进行局部变量的定义，如：

```
foo=0
funct_1(){
       local foo
       foo=1
}
```

函数内部和外部可以有相同名称的变量。它们之间的值互不影响。

8. if 分支语句

```
x=5
if [ $x=5 ]; then
       echo "x equals 5."
else
       echo "x does not equals 5."
fi
```

中括号的两侧，内侧和外侧都需要包含一个空格。

如果需要并列的 if-else，则需要写成 elif，类似于 Java 中的 else if

9. 退出状态

每一条命令执行完毕之后，会向操作系统发出一个值，称为退出状态。这个值是 0-255 之间的数值。0 表示执行成功，其余表示执行失败。使用 `echo $?`可以打印出该值。

shell 提供了两个简单的内置命令，true 和 false。它们唯一的目的是表示命令执行成功（true）或者失败（false）。

10. 测试的条件

选项	含义
file1 -ef file2	file1 和 file2 是否拥有相同的节点编号，即是否通过硬链接指向同一个文件
file1 -nt file2	file1 是否比 file2 新
-d file	file 存在并且是一个目录
-o file	file 存在
-f file	file 存在并且是一个普通文件
-s file	file 存在并且长度大于 0
-w file	file 是否可写
string	string 不为空
-n string	string 长度大于 0

-z string	string 长度等于 0
string1=!string2	string1 不等于 string2
string1==string2 string1=string2	string1 等于 string2
string1>string2	在排序时，string1 是否在 string2 之后
int1 -eq int2	判断整数 int1 是否等于 int2

11. 更现代的 test 版本

    bash 最近的版本增加了一个基于正则表达式的判断。

    语法： `string=~regex`

    例如： `if [[ "$INT"=~^-?[0-9]+$ ]] ; then`

    `[[ ]]`与之前测试中的`[]`作用类似，只是增加了对正则表达式的支持。

12. `(( ))`专为整数而设计的判断

    该操作专门用于整数运算，支持一套完整的算数运算。当算数结果非 0 时，测试为真。

    该命令中因为只用于进行算数计算，所以变量可以直接使用，而无需加上$前缀

```
INT=-5
if ((INT<0)); then
```

13. 组合表达式

    在`[[ ]]`和`(())`中，可使用`&&`， `||`， ！连接多个表达式。

14. read——从标准输入读取值

    如 read int　　#从键盘上读取一个内容，送入变量 int

选项	含义
-a array	将输入值从索引为 0 的位置开始赋值给 array。
-d delimiter	用字符串 delimiter 的第一个字符标志输入的结束，而不是新行的开始
-o	使用 readline 输入
-n num	从输入中读取 num 个字符，而不是一整行
-p prompt	使用 prompt 作为提示，提示用户进行输入
-r	原始模式，不能将后斜线字符翻译为转意码
-s	保密模式，不在屏幕显示输入的字符
-t seconds	超时。在 sceonds 后结束输入。若超时，read 命令返回一个非 0 的退出状态
-u fd	从文件说明符 fd 读取输入，而不是从标准输入中读取

15. while 循环

```
count=1
while [ $count -le 5 ]; do
       echo $count
       count=$((count+1))
done
```

    break 和 continue 用于跳出和继续本次循环。

16. until 循环

while 循环在退出状态不为 0 时循环，until 刚好相反。该循环会在接收到为 0 的退出状态时终止。

```
count=1
until [ $count -ge 5 ]; do
       echo $count
       count=$((count+1))
done
```

17. case 分支

```
read -p "Enter selection [0-3] >"
case $REPLAY in
       0)     echo "Program terminated"
              exit
              ;;
       1)     echo  "Hostname: $HOSTNAME"
              uptime
              ;;
       2)     df -h
              ;;
       3)     ls
              ;;
       *)     echo "Invalid entry" > &2
              exit 1
              ;;
esac
```

case 中的模式都以)结尾的，下面是所有可用的模式

模式	含义
a)	若关键字为 a 则吻合
[[:alpha:]]	若关键字为单个字母则吻合
???)	若关键字为 3 个字符则吻合
*.txt)	若关键字以 txt 结尾则吻合
*)	与任何关键字吻合。此模式应该放在 case 的最后

在 case 中，我们也可以使用|来组合多个模式。模式之间是或的关系。如 q|Q)

18. for 循环

- 传统 shell 形式的 for 循环

  语法：

  ```
  for variable [in words] ; do
     commands
  done
  ```

  例如：

  ```
  for i in A B C D; do
     echo $i
  done
  ```

  现代程序设计语言中用 i, j, k, m 等表示循环变量来自于 Fortran 语言。在该语言中，ijkm 开头的未声明变量自动被归类为整数，而其它字母开头的归类为实数。

- C 语言形式的 for 循环

  ```
  for ((i=0; i<5; i++)); do
     echo $i
  done
  ```

19. 位置参数（命令行参数）

```
echo $0                #我们用$0 表示命令的实际路径名
```

```
echo $1                    #表示用户在该程序执行时输入的第一个参数，以此类推
```

如将上述内容保存到一个脚本中，为 args，执行 `sh args a b`，则得到结果

```
args
a
```

$#用来输出实际参数的个数，如 echo $#

20. shift——处理大量的实际参数

    shell 提供了一种方式，每次执行 shift 时，所有的参数均下移一位(如$2 放入$1..)

```
#! /bin/bash
count=1
while [[ $# -gt 0 ]]; do
       echo "Argument $count = $1"
       count=$((count+1))
       shift
done
```

21. 处理多个位置参数

参数	含义
$*	可扩展为从 1 开始的位置参数列。但包含在双引号内时，扩展为双引号引用的全部位置参数构成的字符串，每个位置参数以空格隔开
$@	可扩展为从 1 开始的位置参数列。当包含在双引号内时，将每个位置参数扩展为用双引号包含起来的单独单词

22. 参数扩展

```
a="foo"
echo "$a_file"       #此处将 a_file 作为变量名，所以没有输出
echo "${a}_file"     #此处得到 foo_file
```

- 空变量扩展的管理

```
foo=
echo ${foo:-"substitute value if unset"}    #:-实际上是 foo 为空则得到后面的默认值
echo foo
foo=bar
echo ${foo:-"substitute value if unset"}
echo $foo
```

-

23. 算数计算和扩展

    在$(( ))进行算数运算时，数值默认为 10 进制，0 开头的是 8 进制，0x 开头的是 16 进制，number#开头的是 number 进制。

    运算符+-*/，**是求幂，%求模

    赋值操作符和++/--都与 C 语言类似

    位操作符：~取反，<<左移，>>右移，&按位与，|按位或，^按位异或

    逻辑操作符与 Java 相同，如<=, >==, !=, ==

    bc——计算命令

24. 数组

- 当访问数组变量时会自动创建一个数组。

a[1]=foo

echo ${a[1]}

使用 declare 命令也可以创建一个数组

declare -a a　　#使用-a 参数创建一个 a 数组

- 数组赋值

除了 a[1]=foo，还可以使用 a=(foo bar)的形式进行赋值。如

```
days=(Sun Mon Tue Wed Thu Fri Sat)
```

还可以通过指定下标进行赋值，如

```
days=([0]=Sun [1]=Mon [2]=Tue [3]=Wed [4]=Thu [5]=Fir [6]=Sat)
```

- 输出数组的内容：使用下标*和@来访问数组的每个元素

```
animals=("a dog" "a cat" "a fish")
for i in "${animals[*]}"; do echo $i; done
for i in "${animals[@]}"; do echo $i; done

for i in ${animals[*]}; do echo $i; done
for i in ${animals[@]}; do echo $i; done
```

- 确定数组元素的数目

```
a[100]=foo
echo ${#a[@]}    #得到结果 1
```

- 查找数组中使用的下标

```
foo=([2]=a [4]=b [6]=c)
for i in "${foo[@]}"; do echo $i; done      #得到 a b c
for i in "${!foo[@]}"; do echo $i; done     #得到 2 4 6
```

- 在数组的结尾增加元素

```
foo+=(d e f)
for i in "${foo[@]}"; do echo $i; done
```

- 数组的排序操作

数组没有直接的方式进行排序，但是可以使用 sort 命令配合管道完成

```
a_sorted=($(for i in "${a[@]}" ; do echo $i; done | sort))
```

- 数组的删除

unset 命令可以删除一个数组，如

```
unset foo      #删除一个数组 foo
```