shell 脚本编程

#!/bin/bash 表示该脚本语言的解释器

两种运行脚本方式：

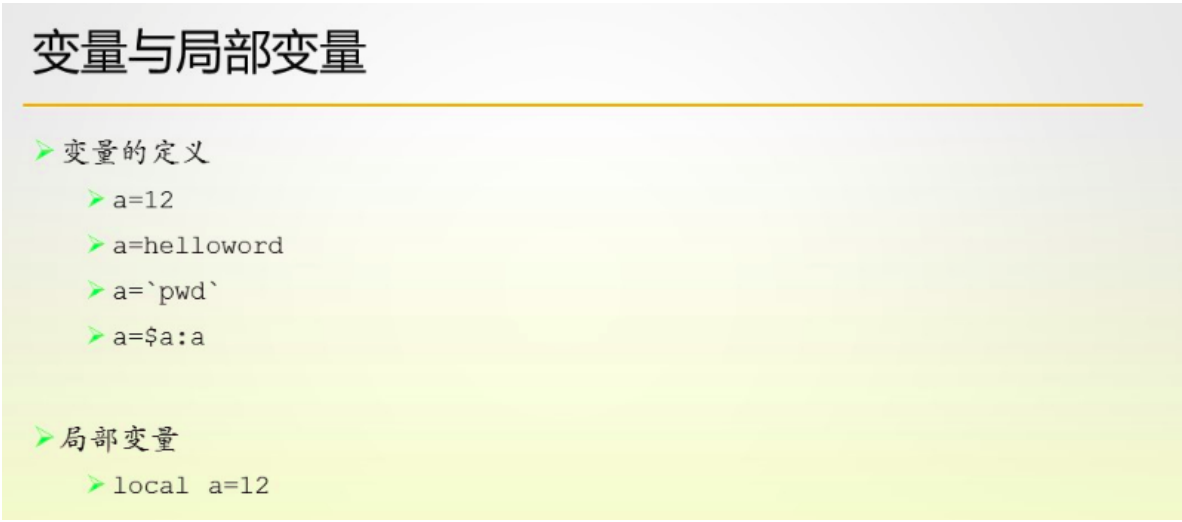
bash filename 运行的是bash脚本文件——filename.sh

（还可以使用source filename来运行文件，之后还可以运行脚本中的单个函数，尽量还是不要使用，一般source主要是，更改了.bashrc、.bash_profile，此时可以source .bashrc 进行更新。主要是用于执行sh文件中的单个函数）

chmod 将脚本变成可执行文件，添加到PATH路径中，直接输入文件名执行。

弱类型和强类型语言 （python不也是吗）

a=12, 至于a是什么类型，要看后面是如何用的



shell编程中存在特殊的位置变量：

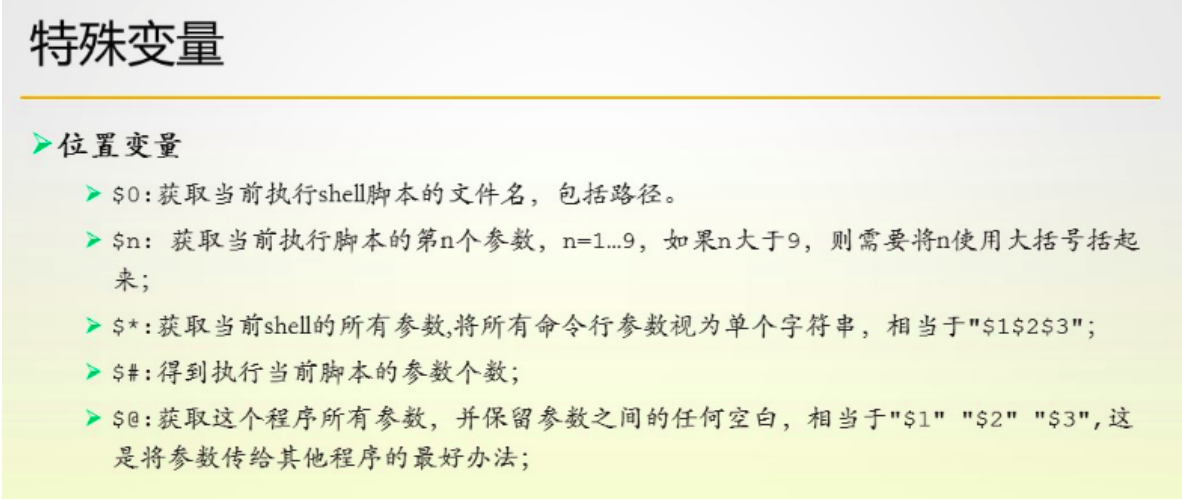
$0: 表示脚本文件的名称

$i：表示脚本文件的第i个参数

$*：表示即脚本文件的所有参数，所有的参数组成一个字符串

$@：同样是将脚本文件的所有参数，不过保留参数之间的空白

$#：求出脚本文件的参数个数，#有求长度和个数的意思

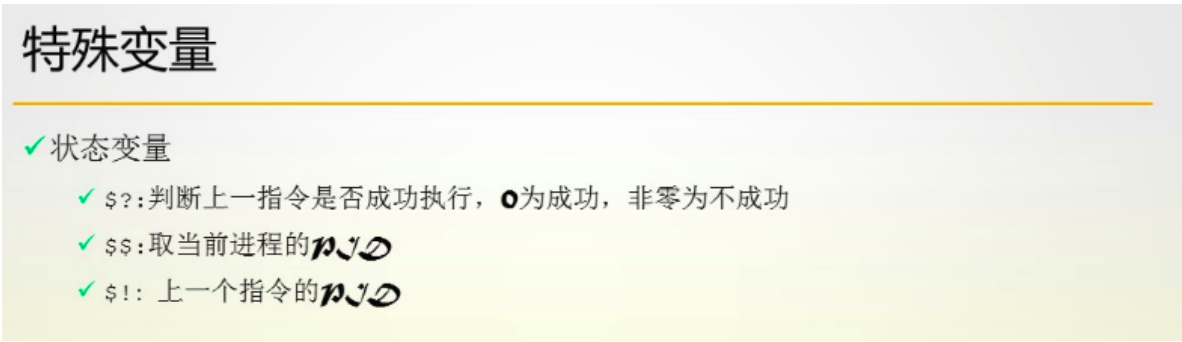


特殊的状态变量：

$？：shell脚本执行的状态，0表示执行成功，非零表示执行失败（联想C语言中main函数返回值为0）

$$：返回脚本对应线程的pid

$!：返回上一条指令的pid



变量与参数的展开，主要是当变量没有定义时，此时整个表达式的值应该是多少？

${var:-word} 当变量var未定义，此时表达式的值为word，变量var仍然处于未定义状态

${var:=word} 当变量var没定义时，此时将变量var的值赋成word，同时表达式的值为var

${var:+word} 当变量定义时，此时返回word

${var:?word} 捕捉因变量没有定义而造成的程序错误

${!var@} 表达式的值为var开头的变量

变量，参数展开

- ✓ ${parameter:-word}如果变量未定义，则表达式的值为**word**。
- ✓ ${parameter:=word}如果变量未定义，则设置变量的值为**word**，返回表达式的值也是**word**。
- ✓ ${parameter:?word}用于捕捉由于变量未定义而导致的错误并退出程序。
- ✓ ${parameter:+word}如果变量已经定义，返回**word**，也就是真。
- ✓ ${!prefix*}
- ✓ ${!prefix@} **prefix**开头的变量

还有关于字符串的一些操作 很重要！！！！

字符串展开

- ✓ **${#parameter}** 输出字符串的长度
- ✓ **${parameter:offset}** 从第**offset**字符开始截取
- ✓ **${parameter:offset:length}** 从**offset**字符开始截取，取**length**长度
- ✓ **${parameter#pattern}** 从头删除最短匹配
- ✓ **${parameter##pattern}** 最长
- ✓ **${parameter%pattern}** 从尾删除最短
- ✓ **${parameter%%pattern}** 从尾删除最长

字符串展开

- ✓ **${parameter/pattern/string}** 第一个匹配被替换
- ✓ **${parameter//pattern/string}** 全部匹配被替换
- ✓ **${parameter/#pattern/string}** 字符串开头的替换
- ✓ **${parameter/%pattern/string}** 字符串结尾的替换
- ✓ **${parameter,,} ${parameter^^}** 全部转换为小写、大写
- ✓ **${parameter,} ${parameter^}** 首字母转换为小写、大写

输入输出的操作

输入输出 - READ

- read [-options] [variable…]
  - -a array　　#把输入赋值到数组 array 中，从索引号零开始。
  - -d delimiter　　#用字符串 delimiter 中的第一个字符指示输入结束，而不是一个换行符
  - -e　　#使用 Readline 来处理输入。这使得与命令行相同的方式编辑输入
  - -n num　　#读取 num 个输入字符，而不是整行
  - -p prompt　　#为输入显示提示信息，使用字符串 prompt
  - -r　　#Raw mode. 不把反斜杠字符解释为转义字符
  - -s　　#Silent mode.
  - -t seconds　　#超时
  - -u fd　　#使用文件描述符 fd 中的输入，而不是标准输入

echo输出

```
echo -e "hello world\n" #若没有-e，就不会输出换行
name="dengruizhi"
echo "hello $name, this is China"
echo "\"dengruizhi\"" #此时将会输出:"dengruizhi",若不加\，此时输出dengruizhi
```

输入输出 - ECHO

- echo string
  - echo -e "Hello HaiZei\n"　　#开启转义
  - echo "Hello $name, This is HaiZei"
  - echo "\"Hello HaiZei\""

还可以想类似c语言中printf 输出

```
printf "hello %s, this is China\n" "dengruizhi" #将进行字符串的替换,同时换行
```

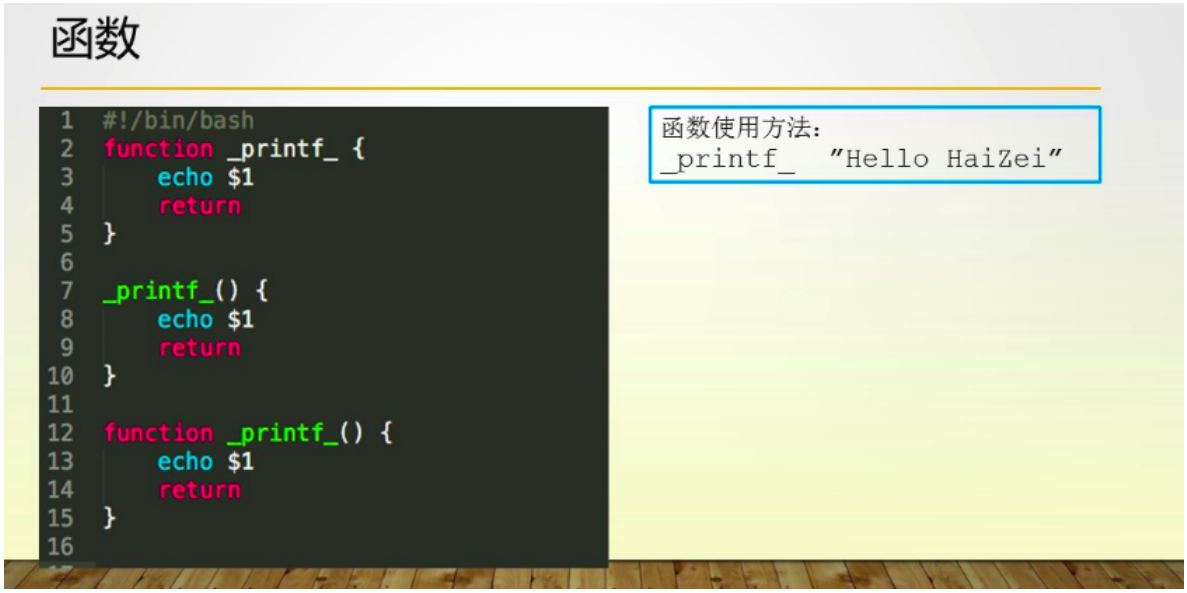
输入输出 - PRINTF

- printf format-string [arguments…]

```
suyelu@HaiZei-Tech:~$ printf "Hello %s,This is HaiZei\n" "Small A"
Hello Small A,This is HaiZei
suyelu@HaiZei-Tech:~$
```

函数的声明以及使用

```
function _print_() {
    #do something, for example: echo $1
    return
}
_print_ "dengruizhi"
#函数名后的参数列表，里面没有参数，只是形势
#此时函数将会echo输出dengruizhi
```



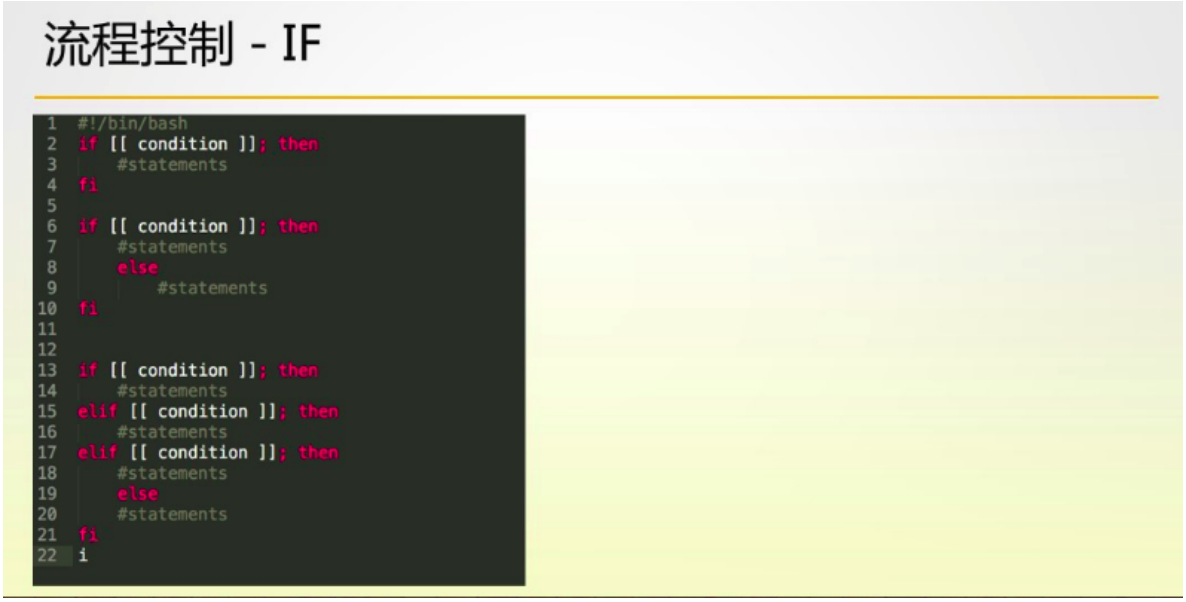
---

if 语句：[[condition]] test表达式 ，可以man test手册打开

test表达式中：在判断string使用=、!=，对于整形的数据使用：equal greater less，还可以判断file文件

```
if [[ condition ]]; then
    #do something
elif [[ condition ]]; then
    #do something
    else
    #do something
fi
```

其中需要注意的是：if两层中括号后面加分号同时then，结束后不要忘记fi



同时对于判定语句，需要了解>,<, >=, <=, ==, !=, 还有与或非 这些基本表示关系的写法

了解字符串的等和不等关系，字符串的长度是否为0

对于整形数据，此时大小关系需要了解

对于文件数据，此时比较的是文件的新旧

这些忘记了都可以询问man手册, 输入man test即可

---

for循环、while循环、until循环

在循环中有和C语言一样的continue、break、若想要退出sh文件的执行: exit

```
while [[ condition ]]; do
    #do something
done
```

```
for i in `seq 0 100`; do
    echo $i
done
```

```
until [[ condition ]]; do
    #do something
done
```

流程控制 - WHILE

```
#!/bin/bash
while [[ condition ]]; do
    #statements
done
```

流程控制FOR

```
for i in words; do
    #statements
done

for (( i = 0; i < 10; i++ )); do
    #statements
done
```

流程控制 - UNTIL

```
#!/bin/bash
until [[ condition ]]; do
    #statements
done
```

数组 重要!!!

数组

- ✓ declare -a a
  - ✓ name[subscript]=value
  - ✓ name=(value1 value2 …)

数组操作

- ✓输出数组内容
  - ✓${arry[*]}
  - ✓${arry[@]}
- ✓确定数组元素个数
  - ✓${#arry[@]}
- ✓找到数组的下标
  - ✓${!arry[@]}

数组操作

- ✓数组追加
  - ✓arry+=(a b c)
- ✓数组排序
  - ✓sort
- ✓删除数组与元素
  - ✓unset

```
#输出数组中的元素
for i in ${arr[@]}; do
    echo $i
done
#数组中一般常用自增操作
let i+=1
i=$[$i+1]
((i++))
#得到sh文件的绝对路径
filepath=$(readlink -f .) #$(readlink -f ..) 得到上上级目录的路径
echo $filepath
```

```
#字符串的截取
str="aabbccssaaddssfg"
echo ${str:2:4} #表示从下标为2的字母开始，向后截取长度为4的字符串，bbcc
echo ${str:3} #若只有一个参数，表示从下标为3的字母开始，一直截取到最后，bbccssaaddssfg
echo ${str:(-2)} #注意截取的方向永远是从左到右，且只有一个参数的时候表达的永远是开始截取下标
#表示从倒数第2的元素开始，向后截取元素
#注意，若是两个参数，后面表示截取长度的值不可以为负数
```

```
#字符串的替换
str="aabbaaccddefgfe"
echo ${str/a/x} #表示将str从左到右，第一个字母a替换成为x，
xabbaaccddefgfe
echo ${str//a/x} #表示将str从左到右，所有的字母a替换成x，
xxbbxxccddefgfe
```

```
echo $RANDOM #产生随机数
```

挂起 Ctrl Z

jobs 看挂起的任务

fg 1：拿到前台执行

bg 1：拿到后台执行

若想查看基本语法man bash查看即可

若对表达式关系不了解 man test查看