1. 描述
   1. Shell脚本：一种为shell编写的脚本程序
   2. Shell环境：只要有一个能编写代码的文编编辑器和一个能解释执行的脚本解释器就可以了
   3. Linux的Shell种类众多，常见的有：
   4. Bourne Shell（/usr/bin/sh或/bin/sh）
   5. Bourne Again Shell（/bin/bash）
   6. C Shell（/usr/bin/csh）
   7. K Shell（/usr/bin/ksh）
   8. Shell for Root（/sbin/sh）
   9. 本教程关注的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数Linux 系统默认的 Shell。
   10. 在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 #!/bin/sh，它同样也可以改为 #!/bin/bash。
   11. #! 告诉系统其后路径所指定的程序即是解释此脚本文件的 Shell 程序
2. 编写并执行shell脚本
   1. 打开文本编辑器(可以使用 vi/vim 命令来创建文件)，新建一个文件 test.sh，扩展名为 sh（sh代表shell），扩展名并不影响脚本执行，见名知意就好，如果你用 php 写 shell 脚本，扩展名就用 php 好了。
   2. 输入一些代码，第一行一般是这样：
      1. #!/bin/bash
      2. echo "Hello World !"
      3. #! 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，即使用哪一种 Shell
      4. echo 命令用于向窗口输出文本
   3. 运行shell脚本有两种方法：
      1. 作为可执行程序
         1. 将上面的代码保存为test.sh，并cd到相应目录
         2. chmod +x ./test.sh #使脚本具有执行权限
         3. ./test.sh #执行脚本
         4. 注意，一定要写成./test.sh，而不是test.sh，运行其他二进制的程序也一样，直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找
      2. 作为解释器参数
         1. 这种运行方式是，直接运行解释器，其参数就是shell脚本的文件名
         2. /bin/sh test.sh
         3. /bin/php test.php
         4. 这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用
3. 变量
   1. your\_name=”runoob” 变量名和等号之间不能有空格：直接赋值
   2. for file in ‘ls /etc’ 或 for file in $(ls /etc)：用语句给变量赋值
   3. 变量名规则
      1. 只能包含字母、数字和下划线，且大小写敏感
      2. 不能以数字开头
      3. 避免使用Shell关键字
      4. 使用大写字母表示常量：PI=3.14
      5. 避免使用特殊符号
      6. 避免使用空格：因为空格通常用于分割命令和参数
   4. 使用变量
      1. your\_name="qinjx"
      2. echo ${your\_name}
   5. 只读变量：readonly myUrl
   6. 删除变量：unset variable\_name
   7. 变量类型
      1. 字符串变量：my\_string='Hello, World!' 或 my\_string="Hello, World!"
         1. 字符串可以用单引号，也可以用双引号，也可以不用引号
         2. 单引号：
            1. 单引号里得任何字符都会原样输出，单引号字符串中得变量是无效的
            2. 单引号字符串中不能出现单独一个的单引号（对单引号使用转义符也不行），但可成对出现，作为字符串拼接使用
            3. 拼接字符串：greeting\_2='hello, '$your\_name' !'

greeting\_3='hello, ${your\_name} !'

echo $greeting\_2 $greeting\_3

* + - 1. 双引号：
         1. 双引号里可以有变量
         2. 双引号里可以出现转义字符
         3. 拼接字符串：greeting="hello, "$your\_name" !"

greeting\_1="hello, ${your\_name} !"

echo $greeting $greeting\_1

* + - 1. 获取字符串长度：string="abcd"

echo ${#string} # 输出 4

或者

string="abcd"

echo ${#string[0]} # 输出 4

* + - 1. 提取子字符串：string="runoob is a great site"

echo ${string:1:4} # 输出 unoo

* + - 1. 查找子字符串：string="runoob is a great site"

echo `expr index "$string" io` # 输出 4

 以上脚本中 **`** 是反引号，而不是单引号 **'**

* + 1. 整数变量：declare -i my\_integer=42
    2. 数组变量：
       1. my\_array=(1 2 3 4 5)：整数索引数组
       2. declare -A associative\_array：关联数组

associative\_array["name"]="John"

associative\_array["age"]=30

* + - 1. 单独定义属组的各个分量：可以不使用连续的下标，而且下标的范围没有限制
         1. array\_name[0]=value0
         2. array\_name[1]=value1
         3. array\_name[n]=valuen
      2. 读取数组
         1. ${属组名[下标]}
         2. valuen=${array\_name[n]}
         3. 使用@符号可以获取数组中的所有元素：echo ${array\_name[@]}
      3. 获取数组的长度
         1. length=${#array\_name[@]}或length=${#array\_name[\*]}
         2. 获取数组单个元素的长度：length=${#array\_name[n]}
    1. 环境变量：echo $PATH
    2. 特殊变量：有一些特殊变量在Shell中具有特殊含义，例如：
       1. $0：表示脚本的名称
       2. $1,$2：表示脚本的参数
       3. $#：表示传递给脚本的参数数量
       4. $?：表示上一个命令的退出状态等
  1. 注释
     1. #：单行注释
     2. 每一行加个#符号太费力了，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果
     3. 多行注释：（: 是一个空命令，用于执行后面的 Here 文档，<<'EOF' 表示开启 Here 文档，COMMENT 是 Here 文档的标识符，在这两个标识符之间的内容都会被视为注释，不会被执行）
        1. :<<EOF

注释内容...

注释内容...

注释内容...

EOF

* + - 1. EOF 也可以使用其他符号:

: <<'COMMENT'

这是注释的部分。

可以有多行内容。

COMMENT

:<<'

注释内容...

注释内容...

注释内容...

'

:<<!

注释内容...

注释内容...

注释内容...

!

* + - 1. 直接使用:号，并用单引号’将多行内容括起来（**: + 空格 + 单引号**）

: '

这是注释的部分。

可以有多行内容。

'

1. 传递参数
   1. 可以在执行脚本的时候，向脚本传递参数，脚本内获取参数的格式为$n，n代表一个数字，1为执行脚本的第一个参数，2为执行脚本的第二个参数
   2. 例如：

**echo** "Shell 传递参数实例！";  
**echo** "执行的文件名：$0";  
**echo** "第一个参数为：$1";  
**echo** "第二个参数为：$2";  
**echo** "第三个参数为：$3";

* 1. 执行：
     1. chmod +x test.sh （为脚本设置可执行权限）
     2. $ ./test.sh 1 2 3 （执行）
  2. 还有几个特殊字符用来处理参数
     1. $#：传递到脚本的参数个数
     2. $\*：以一个单字符串显示所有向脚本传递的参数（如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数）
     3. $$：脚本运行的当前进程ID号
     4. $!：后台运行的最后一个进程的ID号
     5. $@：与$\*相同，但是使用时加引号，并在引号中返回每个参数（如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数）
     6. $-：显示Shell使用的当前选项，与set命令功能相同
     7. $?：显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误
  3. 实例：
     1. **echo** "Shell 传递参数实例！";  
        **echo** "第一个参数为：$1";  
          
        **echo** "参数个数为：$#";  
        **echo** "传递的参数作为一个字符串显示：$\*";
  4. 输出
     1. $ chmod +x test.sh

$ ./test.sh 1 2 3

Shell 传递参数实例！

第一个参数为：1

参数个数为：3

传递的参数作为一个字符串显示：1 2 3

* 1. 

1. 数组
   1. 描述：
      1. 只支持一维数组，初始化时不需要定义数组大小，下标由0开始
   2. 语法格式：
      1. array\_name=(value1 value2 … valuen)
   3. 实例：
      1. my\_array=(A B "C" D)
      2. array\_name[0]=value0

array\_name[1]=value1

array\_name[2]=value2

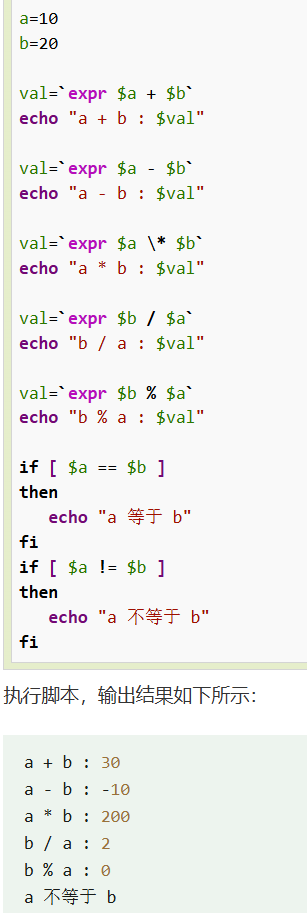
* 1. 读取数组：${array\_name[index]} 如：**echo** "第一个元素为: ${my\_array[0]}"
  2. 关联数组
     1. 描述：
        1. Bash支持关联数组，可以使用任意的字符串、或者整数作为下标来访问数组元素
     2. 语法格式：declare -A array\_name
     3. -A：用于声明一个关联数组
     4. 实例：
        1. declare -A site=(["google"]="www.google.com" ["runoob"]="www.runoob.com" ["taobao"]="www.taobao.com")
        2. declare -A site

site["google"]="www.google.com"

site["runoob"]="www.runoob.com"

site["taobao"]="www.taobao.com"

* + 1. 访问：array\_name[“index”] **echo** ${site["runoob"]}
  1. 获取数组中的所有元素：使用@或\*
     1. echo "数组的元素为: ${my\_array[\*]}"
     2. echo "数组的元素为: ${my\_array[@]}"
  2. 获取数组的所有键：在数组前加一个感叹号!
     1. echo "数组的键为: ${!site[\*]}"
     2. echo "数组的键为: ${!site[@]}"
  3. 获取数组的长度：获取数组长度的方法与获取字符串长度的方法相同
     1. echo "数组元素个数为: ${#my\_array[\*]}"
     2. echo "数组元素个数为: ${#my\_array[@]}"

1. 运算符
   1. 算数运算符（a为10，b为20）
      1. +：加法（`expr $a + $b` 结果为 30）
      2. -：减法（`expr $a - $b` 结果为 -10）
      3. \*：乘法（`expr $a \\* $b` 结果为  200）
      4. /：除法（`expr $b / $a` 结果为 2）
      5. =：赋值（a=$b 把变量 b 的值赋给 a）
      6. ==：相等（[ $a == $b ] 返回 false）
      7. !=：不相等（[ $a != $b ] 返回 true）
      8. 注意：
         1. 乘号\*前边必须加反斜杠（\）才能实现乘法运算
         2. 在MAC中shell的expr语法是：$((表达式))，此处表达式中的”\*”不需要转移符号”\”
      9. 
   2. 关系运算符
   3. 布尔运算符
   4. 字符串运算符
   5. 文件测试运算符
   6. 原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如awk和expr，expr最常用。
   7. expr：是一款表达式计算工具，使用它能完成表达式的求值操作
      1. 如：两个数相加

val=`expr 2 + 2`

echo "两数之和为 : $val"

* + 1. 注意：
       1. 表达式和运算符之间要有空格，例如2+2必须携程2 + 2
       2. 完整的表达式要被**` `**包含，注意这个字符不是常用的单引号，在Esc键下边

1. echo命令
2. printf命令
3. test命令
4. 流程控制
5. 函数
6. 输入/输出重定向
7. 文件包含