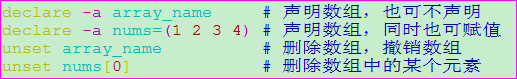
Shell中数据类型不多，比如说字符串，数字类型，数组。数组是其中比较重要的一种，其重要应用场景，可以求数组长度，元素长度，遍历其元素，元素切片，替换，删除等操作，使用非常方便。

Shell中的数组不像JAVA/C，只能是一维数组，没有二维数组;数组元素大小无约束，也无需先定义数组的元素个数;但其索引则像JAVA/C/Python，从0开始，下面其常用的方式进行总结.

【**数组声明**】

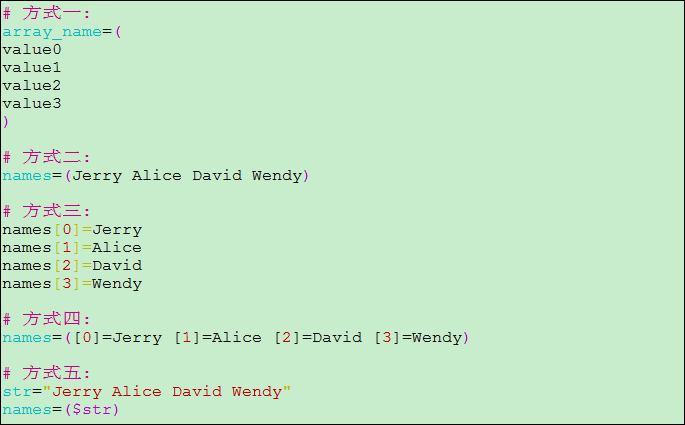


**备注:**

1) 不像JAVA/C等强编程语言，在赋值前必须声明；SHELL只是弱编程语言，可事先声明也可不声明；

2) 用unset来撤销数组，可用unset array\_name[i]来删除里面的元素

【**数组定义**】



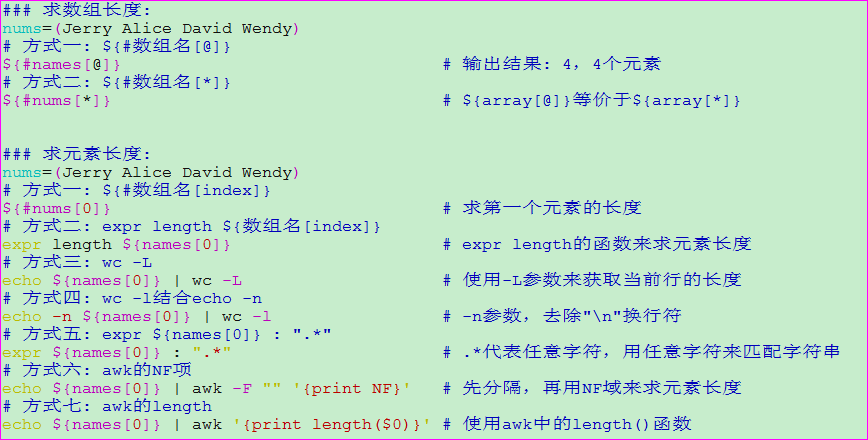
**备注:**

1) 数组中的元素，必须以"空格"来隔开，这是其基本要求；

2) 定义数组其索引，可以不按顺序来定义，比如说:names=([0]=Jerry [1]=Alice [2]=David [8]=Wendy);

3）字符串是SHELL中最重要的数据类型，其也可通过($str)来转成数组，操作起来非常方便；

【**数组长度**】



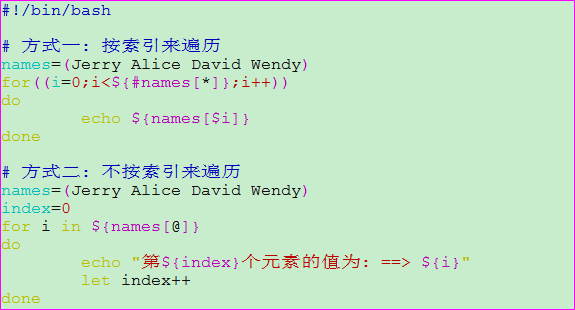
**备注:**

1) 使用${array\_name[@]} 或者 ${array\_name[\*]} 都可以全部显示数组中的元素

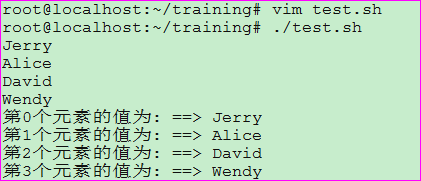
2) 同样道理${#array\_name[@]} 或者 ${#array\_name[\*]}都可以用来求数组的长度

3）求数组中元素的长度方法有很多，相当于求字符串的长度

【**数组遍历**】



脚本输出:

****

**备注:**

1) 可以使用标准的for循环，这种类C语言的方式来遍历数组中的元素

2) for 元素 in 元素集(数组) 这种类Python的方式来遍历数组

3）从代码可读性与执行速度来看，推荐使用第二种方式

【**数组赋值**】

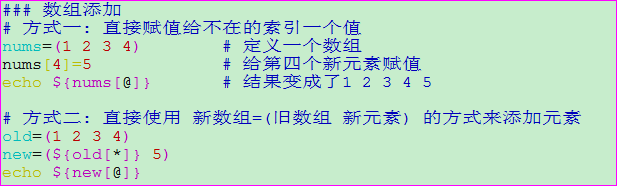
http://img.blog.csdn.net/20160726193205705

**备注:**

1) 第一种是给已经存在的元素项重新赋值

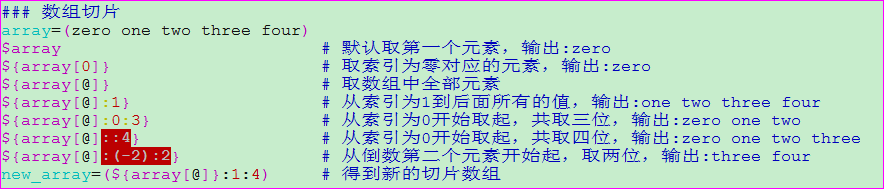
2) 当然也可以给不存在的索引添加赋值，可以看下面的示例

【**数组添加**】

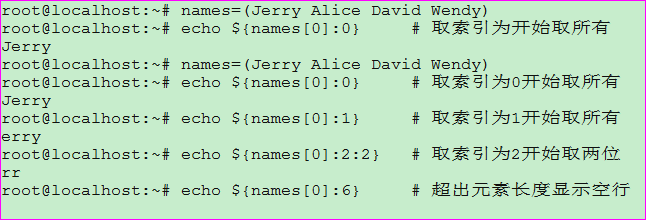


【**数组切片**】

**数组切片**



**元素切片**

****

**备注:**

1) 通用的格式${array[@]:起始位置:长度}，中间以":"隔开，如果第二项省略的话，就取后面所有的项

2) 切片后返回的是字符串，可以通过 新数组=(${旧数组[@]:索引:长度})来索引，参见上面最后一个例子

3) 区别于Python之一:起始位置可以为负数，但必须以放在()中，长度不能为负数

4）区别于Python之二:第二项在Python里面是结束索引，在Shell则代表所取元素的长度

5) 区别于Python之三:Python可以通过 list[-1:-4:-2]来反向取数，在Shell则实现不了

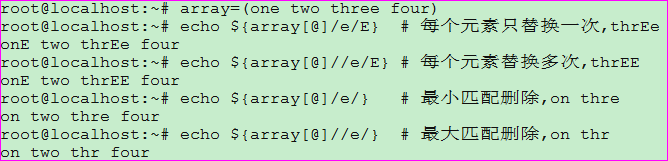
【**数组替换**】

**${array[@]/x/y}**     最小匹配替换，每个元素只替换一次

**${array[@]//x/y}**    最大匹配替换，每个元素可替换多次

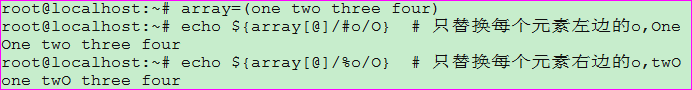
**${array[@]/x/}**      最小匹配删除，只删除一个符合规定的元素

**${array[@]//x/}**     最大匹配删除，可删除多个符合规定的元素



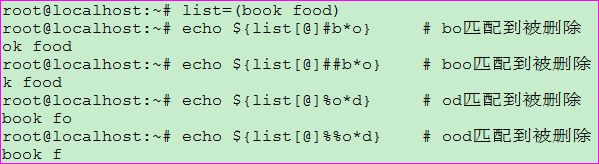
**${array[@]/#x/y}**     从左往右匹配替换，只替换每个元素最左边的字符

**${array[@]/%x/y}**     从右往左匹配替换，只替换每个元素最右边的字符



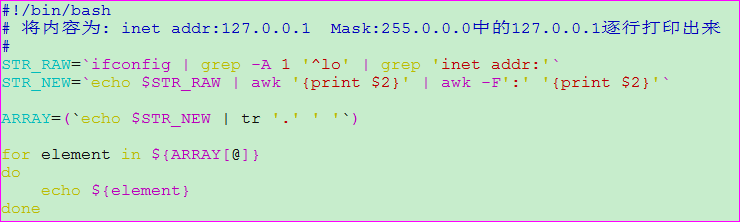
【**数组删除**】

**#**  每个元素,从左向右进行最短匹配  
**##** 每个元素,从左向右进行最长匹配  
**%**  每个元素,从右向左进行最短匹配  
**%%** 每个元素,从右向左进行最长匹配

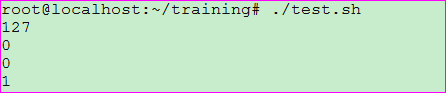


【**数组应用**】

**示例一**: 将ifconfig命令取到的本地IP: 127.0.0.1逐行显示出来



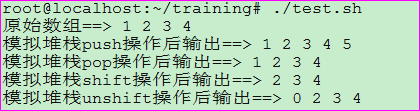
脚本输出:



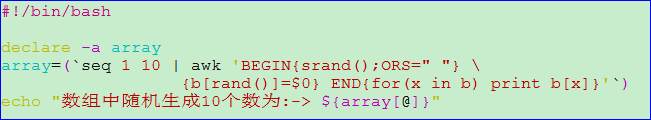
**示例二**: 模拟堆栈的push,pop,shift,unshift操作



脚本输出:



**示例三**: 在1-10间，随机生成10个不重复的数，将其放置于数组中



脚本输出:

http://img.blog.csdn.net/20160727214717729

**备注:**

1) 生成[1,10]范围内不重复的随机整数，并保存到数组array中

2) seq 1 10 用于生成1~10的整数序列(包含边界值1和10)

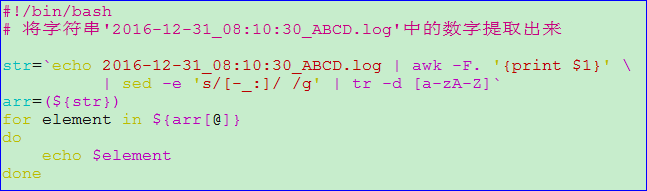
3) awk中的rand()函数用于随机产生一个0到1之间的小数值(保留小数点后6位)

4）rand()只生成一次随机数，要使用srand()函数使随机数滚动生成

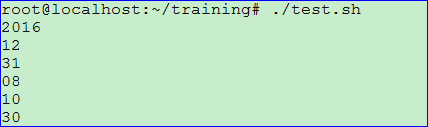
5) 括号里留空即默认采用当前时间作为随机计数器的种子,这样以秒为间隔，随机数就能滚动随机生成了

6) 由于以秒为间隔，所以如果快速连续运行两次脚本(1s内)，你会发现生成的随机数还是一样的

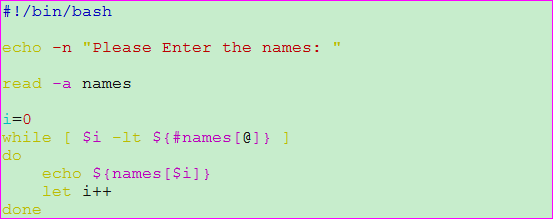
**示例四**: 将字符串处理后转为为数组，再对其打印输出



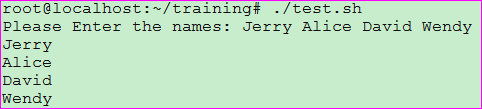
脚本输出:



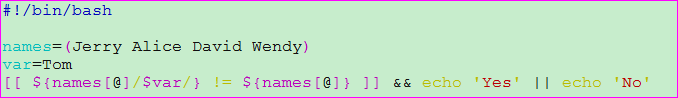
**示例五**: 用read -a参数，从标准输入中读取数组，再做操作



脚本输出:



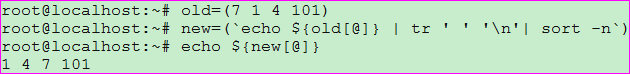
**示例六**:  判断某个变量，是否在数组中，在输出YES，否输出NO



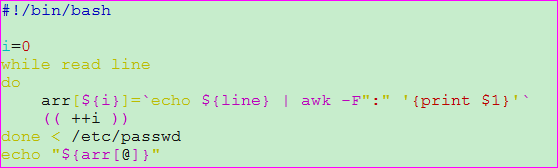
脚本输出:

http://img.blog.csdn.net/20160728123430542

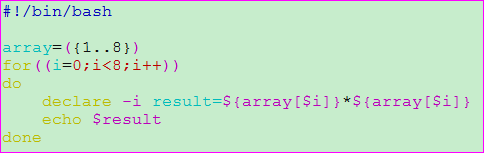
**示例七**:  对数组中的元素进行排序



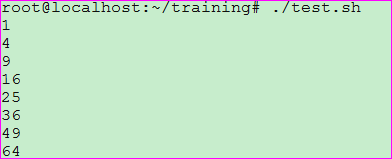
**示例八**:  将/etc/passwd文件中以:分隔的第一列，即用户名放置于一个数组中



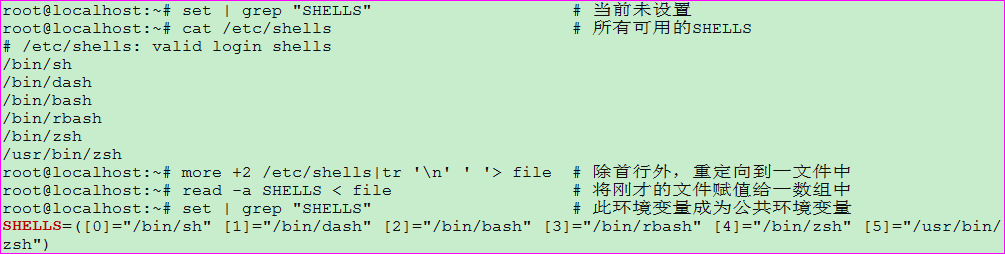
**示例九**:  将1-8，每个数自乘后输出



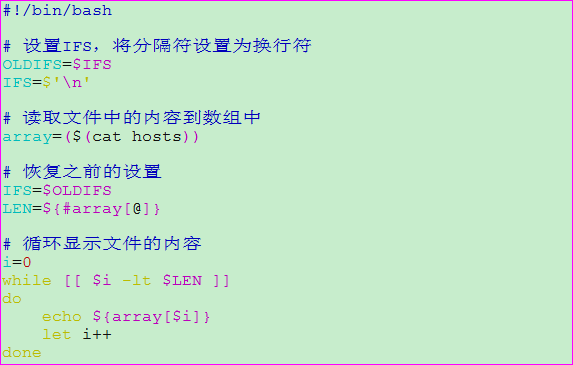
脚本输出:



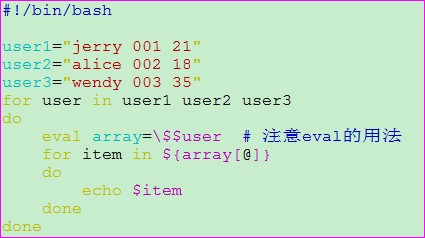
**示例十**:  借助数组来设置SHELLS的环境变量



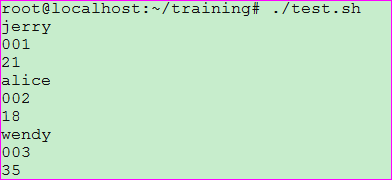
**示例十**一:  设置IFS，读取文件内容示例



**示例十**二:  利用eval，模拟实现数组的功能

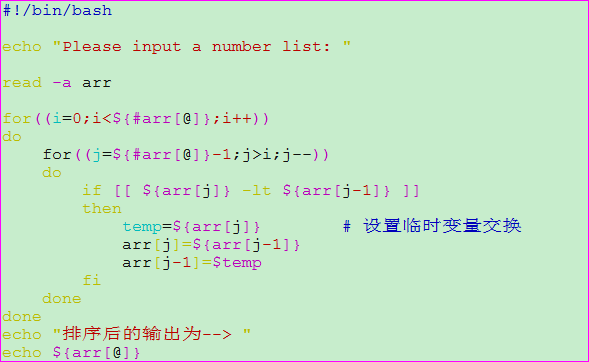


脚本输出:

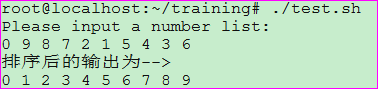


**示例十**三:  利用数组来实现冒泡排序

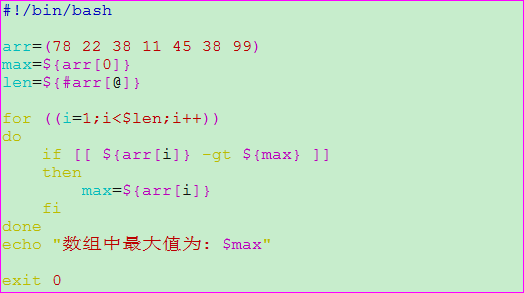
思路:会重复地走访过要排序的数组，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。越大的元素会经由交换慢慢“浮”到数列的顶端



脚本输出:



**示例十四**:  利用数组来求最大值



脚本输出

http://img.blog.csdn.net/20160808101036200