什么是Shell脚本?

将指令放到一个文件中统一执行，且可以设置执行的逻辑。

```
rm -rf AA;mkdir AA;cd AA;touch test{1,2,3}.txt
```

在shell中，将多个指令用';'拼接起来，就是一个指令集合，按照顺序执行。然而，如果命令特别复杂，写在同一行是很不方便的，而且也无法保存命令，所以，我们将指令集合放到一个文件当中保存，统一执行，这就是shell脚本。

这样，我们就可以将工作中重复的流程性操作统一写好脚本，统一执行。

```
#! /bin/bash

rm -rf AA
mkdir AA
cd AA
touch test0{1,2,3}.txt
```

查看在线用户

使用 w 或 who 命令都可以查看服务器上目前已登录的用户信息，两者的区别在于，w 命令除了能知道目前已登陆的用户信息，还可以知道每个用户执行任务的情况。

```
#! /bin/bash

date
w
who
```

脚本执行方法:

1.bash 文件名

2.chmod +x 文件名 ， 再 ./文件名 执行

echo和变量

- 双引号"" :会把引号的内容当成整体来看待，允许通过$符号引用其他变量值
- 单引号'' :会把引号的内容当成整体来看待，禁止引用其他变量值，shell中特殊符号都被视为普通字符
- 反撇号``:反撇号和$()一样，引号或括号里的命令会优先执行，如果存在嵌套，反撇号不能用

```
#! /bin/bash

echo hello cat.
echo "hello cat."
echo 'hello cat.'

```

```
name="blackcat"

echo "hello $blackcat."
echo "hello ${blackcat}."
echo 'hello ${blackcat}.'

echo $(date +%F)
echo `date +%F`
```

只读变量

```
#! /bin/bash

name="cat"
gender="m"

readonly name
declare -r gender

name="blackcat"
gender="f"
```

删除变量

```
#! /bin/bash

name="cat"

echo ${name}

unset name

echo ${name}

echo "done!"
```

变量分类

本地变量

当前用户自定义的变量。当前进程中有效，其他进程及当前进程的子进程无效。

```
hioier@yunpc:~$ name="cat"
hioier@yunpc:~$ echo ${name}
cat
hioier@yunpc:~$ ps
    PID TTY          TIME CMD
  29714 pts/2    00:00:00 bash
  30435 pts/2    00:00:00 ps
hioier@yunpc:~$ bash
hioier@yunpc:~$ echo ${name}

hioier@yunpc:~$ ps
    PID TTY          TIME CMD
  29714 pts/2    00:00:00 bash
  30436 pts/2    00:00:00 bash
  30442 pts/2    00:00:00 ps
```

环境变量

当前进程有效，并且能够被子进程调用。

查看当前用户的环境变量：env
查询当前用户的所有变量（临时变量与环境变量）：set
export：将当前变量变成环境变量

```
hioier@yunpc:~$ export name="cat"
hioier@yunpc:~$ echo ${name}
cat
hioier@yunpc:~$ bash
hioier@yunpc:~$ echo ${name}
cat
```

```shell
# 将普通变量导出为环境变量
declare -x name

# 将环境变量变为普通变量
declare +x name
```

全局变量

全局所有的用户和程序都能调用，且继承，新建的用户也默认能调用。

所有用户的环境变量：/etc/profile

当前用户的环境变量：~/.bashrc

修改后执行：source .bashrc，使环境变量生效。

内置变量

变量	含义
$0	当前脚本的文件名。
$n（n≥1）	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1个参数是1，第2个参数是1，第2个参数是2，第10个参数是 ${10}。
$#	传递给脚本或函数的参数个数。
$*	脚本后面所有参数，参数当成一个整体输出。
$@	脚本后面所有参数，参数是独立的，也是全部输出。
$?	上个命令的退出状态，或函数的返回值。若退出状态值为0，表示命令运行成功。
$$	当前 Shell 进程 ID。对于 Shell 脚本，就是这些脚本所在的进程 ID。

整型变量

```
hioier@yunpc:~/scripts$ declare -i A=10
hioier@yunpc:~/scripts$ declare -i B=20
hioier@yunpc:~/scripts$ declare -i C=$A+$B
hioier@yunpc:~/scripts$ echo $C
30
```

数组变量

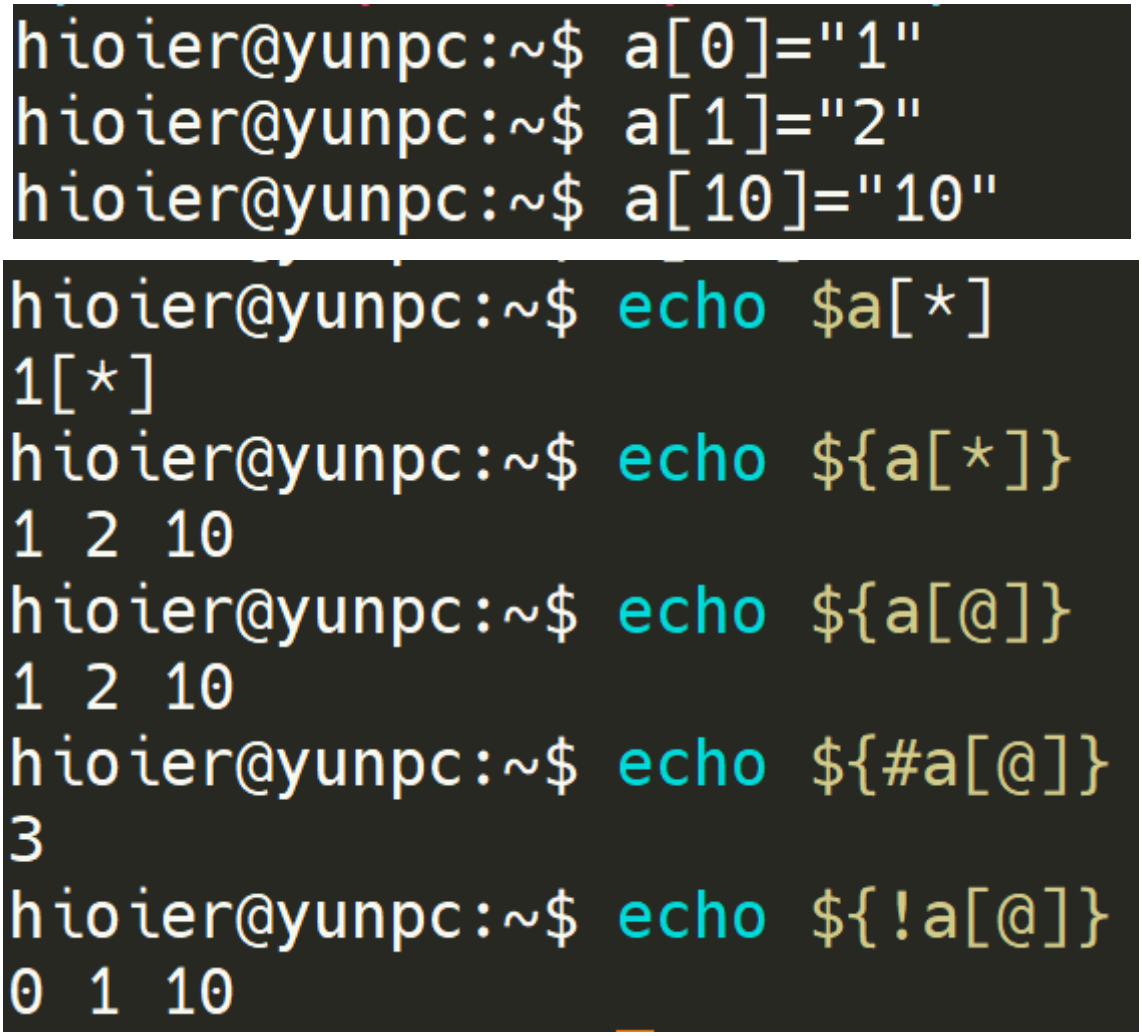
数组中可以存放多个值。Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小。

与大部分编程语言类似，数组元素的下标由 0 开始。

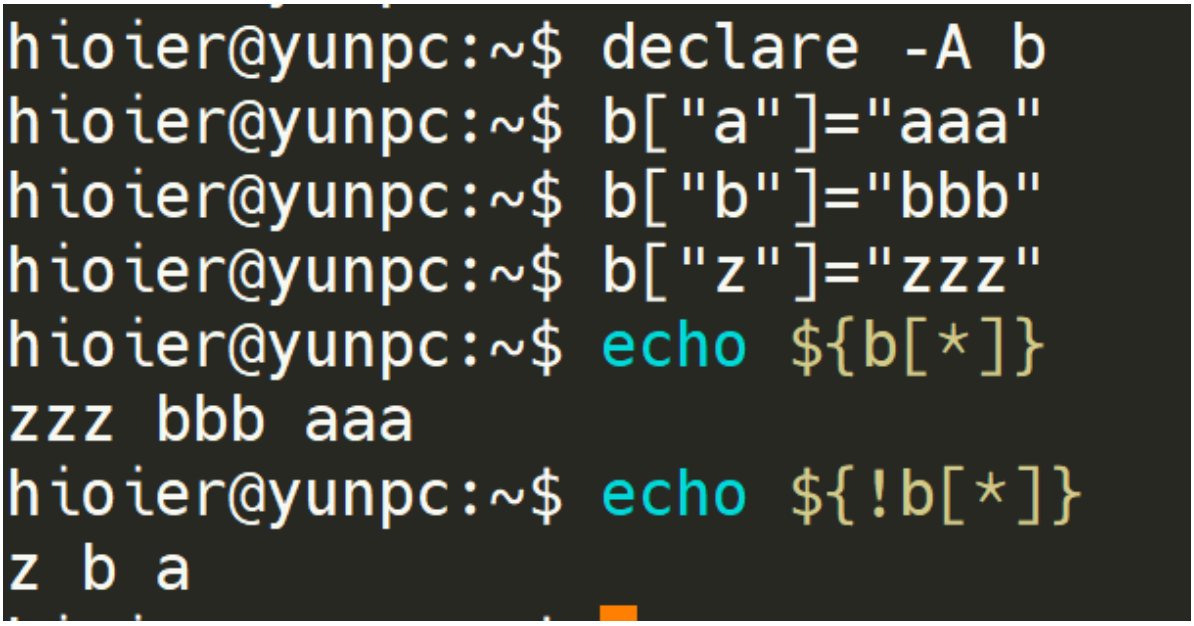
Shell 数组用括号来表示，元素用"空格"符号分割开，语法格式如下：

```
array_name=(value1 value2 ... valuen)
```

普通数组



关联数组



read交互式输入

选项	说明
-a array	把读取的数据赋值给数组 array，从下标 0 开始。
-d delimiter	用字符串 delimiter 指定读取结束的位置，而不是一个换行符（读取到的数据不包括 delimiter）。
-n num	读取 num 个字符，而不是整行字符。
-p prompt	显示提示信息，提示内容为 prompt。
-r	原样读取（Raw mode），不把反斜杠字符解释为转义字符。
-s	静默模式（Silent mode），不会在屏幕上显示输入的字符。当输入密码和其它确认信息的时候，这是很有必要的。
-t seconds	设置超时时间，单位为秒。如果用户没有在指定时间内输入完成，那么 read 将会返回一个非 0 的退出状态，表示读取失败。

数学运算

```
1. 使用 $(( ))
2. 使用 $[ ]
3. 使用 expr 外部程式
4. 使用 let 命令
```

```
hioier@yunpc:~$ echo $((1+2))
3
hioier@yunpc:~$ echo $((1*2))
2
```

```
hioier@yunpc:~$ echo $[1+2]
3
hioier@yunpc:~$ echo $[1*2]
2
```

```
expr表达式说明：

用空格隔开每一项
用反斜杠放在shell特定的字符前面（发现表达式运行错误时，可以试试转义）
对包含空格和其他特殊字符的字符串要用引号括起来
expr会在stdout中输出结果。如果为逻辑关系表达式，则结果为真，stdout为1，否则为0。
expr的exit code：如果为逻辑关系表达式，则结果为真，exit code为0，否则为1。
length STRING：返回STRING的长度
index STRING CHARSET：CHARSET中任意单个字符在STRING中最前面的字符位置，下标从1开始。如果在STRING中完全不存在CHARSET中的字符，则返回0。
substr STRING POSITION LENGTH：返回STRING字符串中从POSITION开始，长度最大为LENGTH的子串。如果POSITION或LENGTH为负数，0或非数值，则返回空字符串。
逻辑关系表达式
|:如果第一个参数非空且非0，则返回第一个参数的值，否则返回第二个参数的值，但要求第二个参数的值也是非空或非0，否则返回0。如果第一个参数是非空或非0时，不会计算第二个参数。
&:如果两个参数都非空且非0，则返回第一个参数，否则返回0。如果第一个参为0或为空，则不会计算第二个参数。

< <= = == != >= >
比较两端的参数，如果为true，则返回1，否则返回0。”==”是”=”的同义词。”expr”首先尝试将两端参数转换为整数，并做算术比较，如果转换失败，则按字符集排序规则做字符比较。

() 可以改变优先级，但需要用反斜杠转义
```