Shell Cheat Sheet

Shell简介

Shell本身是一个用C语言编写的程序，它是用户使用Unix/Linux的桥梁。
Shell脚本和编程语言很相似，也有变量和流程控制语句，但Shell脚本是解释执行的，不需要编译，Shell程序从脚本中一行一行读取并执行这些命令，相当于一个用户把脚本中的命令一行一行敲到Shell提示符下执行。
Shell的两种执行命令方式：
交互式（Interactive）：解释执行用户的命令，用户输入一条命令，Shell就解释执行一条。
批处理（Batch）：用户事先写一个Shell脚本(Script)，其中有很多条命令，让Shell一次把这些命令执行完，而不必一条一条地敲命令。

编译型语言
很多传统的程序设计语言，例如C++和Java，都是编译型语言。这类语言需要预先将我们写好的源代码(source code)转换成目标代码(object code)，这个过程被称作“编译”。
运行程序时，直接读取目标代码(object code)。由于编译后的目标代码(object code)非常接近计算机底层，因此执行效率很高，这是编译型语言的优点。
但是，由于编译型语言多半运作于底层，所处理的是字节、整数、浮点数或是其他机器层级的对象，往往实现一个简单的功能需要大量复杂的代码。例如，在C++里，就很难进行“将一个目录里所有的文件复制到另一个目录中”之类的简单操作。
解释型语言
解释型语言也被称作“脚本语言”。执行这类程序时，解释器(interpreter)需要读取我们编写的源代码(source code)，并将其转换成目标代码(object code)，再由计算机运行。因为每次执行程序都多了编译的过程，因此效率有所下降。

执行Shell脚本

第一行：#!/bin/bash
“#!” 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。
运行Shell脚本有两种方法。

作为可执行程序

将上面的代码保存为test.sh，并 cd 到相应目录：
chmod +x ./test.sh #使脚本具有执行权限
./test.sh #执行脚本
注意，一定要写成./test.sh，而不是test.sh。运行其它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。

作为解释器参数

直接运行解释器，其参数就是shell脚本的文件名
/bin/sh test.sh
/bin/php test.php

Shell变量

变量名和等号之间不能有空格
myNum=100
myUrl="http://see.xidian.edu.cn/cpp/linux/"
使用一个定义过的变量，只要在变量名前面加美元符号（$）即可，如
your_name="mozhiyan"
echo $your_name
echo ${your_name}
推荐给所有变量加上花括号，这是个好的编程习惯。
已定义的变量，可以被重新定义， 第二次赋值的时候不能用$引用原变量，使用变量的时候才加美元符
使用 unset 命令可以删除变量

变量类型

局部变量
局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量。
环境变量
所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量。
shell变量
shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行

Shell命令替换

命令替换是指Shell可以先执行命令，将输出结果暂时保存.
DATE=`date` 先执行该命令， 结果保存到变量DATE里
echo "Date is $DATE”

Shell运算

算术运算符

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。
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。
val=`expr 2 + 2`
注意：
1. 表达式和运算符之间要有空格，例如 2+2 是不对的，必须写成 2 + 2
2. 完整的表达式要被 ` ` 包含
a=10
b=20
val=`expr $a + $b`
val=`expr $a \* $b` 乘号(*)前边必须加反斜杠(\)才能实现乘法运算
也可以用 echo $((1 + 1)), echo $((1 * 1)). a=$((1 + 1))这样更方便而且中间有没有空格无所谓

关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

条件表达式要放在方括号之间，并且要有空格，例如 [$a==$b] 是错误的，必须写成 [$a == $b]

```
if [ $a -eq $b ]
then
   echo "$a -eq $b : a is equal to b"
else
   echo "$a -eq $b: a is not equal to b"
fi
```

常用运算符：-eq, -ne, -gt, -lt, -ge, -le都是英文缩写

布尔运算符

三种：

！ 非运算　　[! false] 返回 true。

-o 或运算　 [$a -lt 20 -o $b -gt 100]返回 true。

-a 与运算　 [$a -lt 20 -a $b -gt 100] 返回 false。

字符串运算符

=　 检测两个字符串是否相等，数字的比较是==

!=

-z 检测字符串长度是否为0，为0返回 true。

-n 检测字符串长度是否为0，不为0返回 true。

str 检测字符串是否为空，不为空返回 true。

文件测试运算符

检测文件各种属性

Shell注释

sh里没有多行注释，只能每一行加一个#号。

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。

Shell字符串

单引号

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的

单引号字串中不能出现单引号（对单引号使用转义符后也不行

双引号

双引号里可以有变量

双引号里可以出现转义字符

获取字符串长度

```
string="abcd"
echo ${#string}  #输出 4
```

提取子字符串

```
string="alibaba is a great company"
echo ${string:1:4}  #输出liba
```

查找子字符串

```
string="alibaba is a great company"
echo `expr index "$string" is`
```

Shell数组

bash支持一维数组（不支持多维数组）

定义数组

在Shell中，用括号来表示数组，数组元素用“空格”符号分割开。array_name=(value1 ... valuen)

还可以单独定义数组的各个分量：array_name[0]=value0

使用 * 可以获取数组中的所有元素　${array_name[*]}

获取数组的长度

```
length=${#array_name[*]}
```

Shell echo

显示结果重定向至文件

```
echo "It is a test" > myfile
```

显示命令执行结果

```
echo `date`
```

Shell逻辑语句

if 语句

```
if [ $a == $b ]
then
  echo "a is equal to b"
fi
```

if else语句

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

if elif else语句

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

Shell循环语句

一般格式：

```
for 变量 in 列表
do
    command1
    command2
    ...
    commandN
done
```

例如

```
for loop in 1 2 3 4 5
do
   echo "The value is: $loop"
done
```

读取文件

```
for FILE in $HOME/.bash*
do
  echo $FILE
done
```

Shell函数

Shell 函数的定义格式如下：

```
function_name () {
    list of commands
    [ return value ]
}
```

Shell 函数返回值只能是整数，调用函数只需要给出函数名，不需要加括号。

在Shell中，调用函数时可以向其传递参数。在函数体内部，通过 $n 的形式来获取参数的值，例如，$1表示第一个参数，$2表示第二个参数...

```
1.  #!/bin/bash
2.  funWithParam(){
3.      echo "The value of the first parameter is $1 !"
4.      echo "The value of the second parameter is $2 !"
5.      echo "The value of the tenth parameter is $10 !" // 不能得到第十个参数
6.      echo "The value of the tenth parameter is ${10} !" // 这样才可以得到
7.      echo "The value of the eleventh parameter is ${11} !"
8.      echo "The amount of the parameters is $# !"  # 参数个数
9.      echo "The string of the parameters is $* !"  # 传递给函数的所有参数
10. }
11. funWithParam 1 2 3 4 5 6 7 8 9 34 73
```

```
The value of the first parameter is 1 !
The value of the second parameter is 2 !
The value of the tenth parameter is 10 !
The value of the tenth parameter is 34 !
The value of the eleventh parameter is 73 !
The amount of the parameters is 12 !
The string of the parameters is 1 2 3 4 5 6 7 8 9 34 73 !"
```

注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。

Shell重定向

输出重定向

命令的输出不仅可以是显示器，还可以很容易的转移向到文件，这被称为输出重定向。
command > file　这样，输出到显示器的内容就可以被重定向到文件。
如果不希望文件内容被覆盖，可以使用 >> 追加到文件末尾

输入重定向

command < file
这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到 /dev/null：
command > /dev/null
起到”禁止输出“的效果。