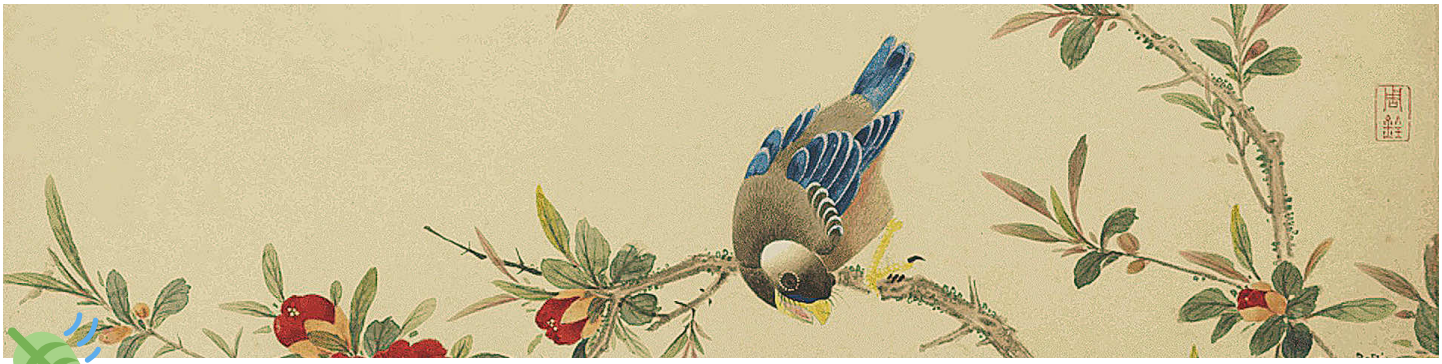


Shell脚本

标签　　空

新增属性

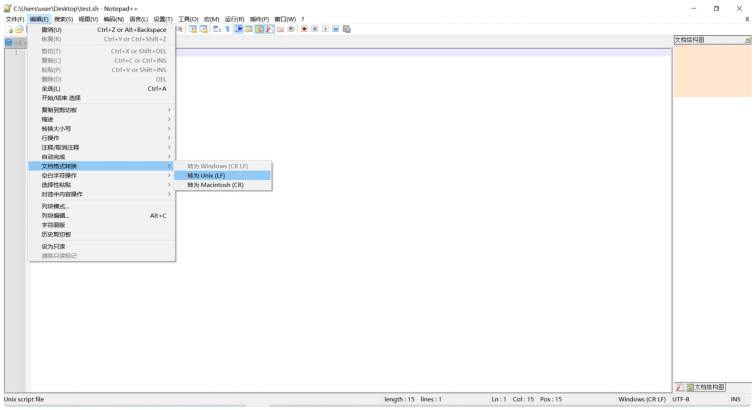
Shell 脚本

shell脚本是Linux上的批处理程序，可以批量执行Linux命令，windows上对应的则是bat脚本。

> 反引号`可包含命令

开始：

在windows上创建一个test.sh后缀的文本文件，并将格式改为Unix格式。



编写完成后，将test.sh文件上传到Linux系统，然后为脚本添加权限：

```PowerShell
# 为指定脚本设置权限
chmod 777 ./test.sh

# 为所有脚本设置权限
chmod 777 ./*.sh
```

执行shell脚本，进入sh所在目录执行如下命令：

```PowerShell
# 不能直接使用test.sh,因为无法识别为命令
./test.sh
```

输入、输出

输入

```PowerShell
read name
echo 输入了$name
```

输出

echo：

```PowerShell
echo aaa

echo "aaa"

echo "${name}aaa"

echo -e"aaa\n"    # -e转义，\n换行，一般自动换行

echo -e"aaa\c"    # -e转义，\c取消换行
echo "bbb"


echo `date`      # 显示date命令的结果
```

printf：和c语言类似

```PowerShell
printf "Hello, Shell\n"
printf "%d %s\n" 1 "abc"
```

变量

- 局部变量：范围在shell脚本、命令内。
- 环境变量：所有程序都能访问。
- shell变量：shell程序设置的变量，一部分是局部变量，一部分是环境变量。

定义变量：变量名和“=”之间，不能有空格。

```PowerShell
name="tom"
echo "hello!${name}"
echo "hello!${name}"
# 使变量只读
readonly name
```

> 结果：hello!tom

使用变量：下面几种方式效果一样。

```PowerShell
echo $name
echo ${name}
```

> 加花括号是为了解释器可以识别变量边界

变量赋值：直接重写变量即可。

```PowerShell
name="demo"
name="emp"
sshs=`rpm -qa | grep openssh`  # 命令结果
sshs=`rpm -qa | grep ${name}`  # 命令结果
echo $name
```

> 结果：emp

变量删除：

```PowerShell
name="demo"
unset name
echo $name
```

> 结果：没有任何输出

数据类型

字符串

无引号、单引号、双引号都能表示字符串

- 单引号：不识别变量、只能成对出现
- 双引号：识别变量，可以有转义字符

```PowerShell
name1="demo"
name2='demo'
# 拼接
name="zhangsan""hello"
```

字符串常用方法：

用途	语法	备注
取文本长度	${#name}	返回长度
截取文本	${name:1:4}	0代表第一个字符
	`awk '{print $2}'字符串`	$2表示第二个，可以使用管道符|查找前一个命令的结果,注意用`包含
查找字符a或b	`expr index "$name" ab`	加`号包含，不是单引号

数组

```PowerShell
# 定义数值
names=(a b c d e)
# 数组赋值
names[0]=a1
# 获取值
echo ${#names[0]}
echo ${#names[@]}
```

> 结果1：a1
> 结果2：a1 b c d e

添加元素

```PowerShell
names[9]=f    # 直接添加，如果已存在则覆盖
```

数组常用方法：

用途	语法	备注
获取元素长度	${names[0]}	返回长度
获取元素个数	$names[@]、{names[*]}	返回个数

注释

单行

```PowerShell
# ----------------------
```

> 也可以把要注释的内容使用 { } 定义成函数，函数未被调用就相当于注释了。

多行：xxx 可以为任意字符。

```PowerShell
:<<XXX
--------------------------
--------------------------
--------------------------
XXX
```

参数传递

> 执行脚本时参数和函数参数用法一致。

执行时传入参数：

```PowerShell
./test.sh 1 2 3
```

内部使用：$0 是文件名，其它为参数值。

```PowerShell
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";
echo "第四个参数为：$4";
echo "第十个参数为：${10}"
```

> $只能自动识别1位，多位需要使用 { }

常用方法：

含义	用法	备注
$#	参数个数	
$*	打印所有参数	字符串形式"a b c"
$@	打印所有参数	字符串形式"a" "b" "c"
$$	脚本进程ID	
$!	最后一个进程ID	与set命令功能相同
$–	Shell正在使用的选项	
$?	最后命令的退出状态	0表示无错

运算

使用`进行包围，并使用expr或 $[ ]进行

```PowerShell
v1=`expr 3+2`  v1=$[3+2]
v2=`expr 3-2`  v2=$[a-b]
v3=`expr 3*2`
v4=`expr 3/2`
v5=`expr 3%2`
```

> https://www.runoob.com/linux/linux-shell-basic-operators.html

关系运算：不支持字符串

```PowerShell
# a=b?
[$a -eq $b]
# a!=b?
[$a -ne $b]
# a>b?
```

```
[$a -gt $b]
# a<b?
[$a -lt $b]
# a>=b?
[$a -ge $b]
# a<=b?
[$a -le $b]
```

字符串运算：

PowerShell

```
# a等于b?
[$a == $b]
# a不等于b?
[$a != $b]
# a长度为0?
[-z $a]
# a长度不为0?
[-n $a]
# a为空?
[$a]
```

文件检测

[操作符][文件名]

操作符	
-b	是否为块设备文件
-c	是否为字符设备文件
-d	是否是目录
-e	目录是否存在
-f	是否是普通文件
-g	文件是否设置了 SGID 位
-k	文件是否设置了粘着位(Sticky Bit
-p	文件是否是有名管道
-u	文件是否设置了 SUID 位
-r	文件是否可读
-w	文件是否可写
-x	文件是否可执行
-s	文件大小是否大于0

流程控制

if...else

PowerShell

```
a=1
b=2
if [ $a != $b ]
then
  echo "--------------"
else
  echo "..............."
fi
```

可以写成一行，用；分隔

```PowerShell
if [ $a != $b ]; then echo "--------------"; else echo "..............."; fi
```

if常和test结合：

```PowerShell
a=1;b=2
if test $a -eq $b
then
  echo "..............."
fi
```

for

```PowerShell
emps=(1 2 3 4)
for emp in $emps
do
  echo $emp
done
```

或

```PowerShell
for emp in 1 2 3 4
do
  echo $emp
done
```

或

```PowerShell
sshs=`rpm -qa | grep openssh`
for ssh in $sshs
do
  echo $ssh
done
```

while

```PowerShell
while 1 2 3 4
do
    command
done
```

无限循环

```PowerShell
while :   ; do echo 666;done
# 或
while true; do echo 666;done
# 或
for (( ; ; )); do echo 666;done
```

> while条件为true执行，until循环条件为true停止

case...esac

```PowerShell
read id
case $id in
  1) echo "demo1"
  ;;
  2) echo "demo2"
  ;;
  3) echo "demo3"
  ;;
esac
```

- break 跳出循环
- continue 跳过循环

函数

> 先写函数，才能调用。

定义函数

```PowerShell
test(){
  name="hello"
  return $name
}
```

调用函数

```PowerShell
test
echo "结果=$?"
```

参数传递

```PowerShell
#定义
test(){
  echo "第一个参数为 $1"
  echo "第二个参数为 $2 !"
  echo "第十个参数为 ${10} !"
  echo "第十一个参数为 ${11} !"
  echo "参数总数有 $# 个!"
  echo "作为一个字符串输出所有参数 $* !"
}
#调用
test 1 2 3 4 5 6 7 8 9 78 67
```

文件操作

修改文件内容

```PowerShell
输入命令 < file  #将输入存到文件

命令 > test  # 将命令结果存到test文件
命令 >> test  # 将命令结果追加到test文件

命令 > /dev/null  # 执行命令，禁止输出结果
```

修改内容：

```PowerShell
sed -i 's/aaa/bbb/g' ./test
# 插入变量,其中/可用#、|替换，如果字符包含[ ],需要转义，[ ]和正则的一样
name="bbb"
sed -i "s/aaa/$name/g" ./test

# 注意由于# | / , 空格 等都能作为分界符，所以替换的文本中要注意
```

> sed元字符

```纯文本
^ 匹配行开始，如：/^sed/匹配所有以sed开头的行;

$ 匹配行结束，如：/sed$/匹配所有以sed结尾的行;

. 匹配一个非换行符的任意字符，如：/s.d/匹配s后接一个任意字符，最后是d;

* 匹配0个或多个字符，如：/*sed/匹配所有模板是一个或多个空格后紧跟sed的行;

[] 匹配一个指定范围内的字符，如/[ss]ed/匹配sed和Sed;
```

[^] 匹配一个不在指定范围内的字符，如：/[^A-RT-Z]ed/匹配不包含A-R和T-Z的一个字母开头，紧跟ed的行;

\(..\) 匹配子串，保存匹配的字符，如s/\(love\)able/\1rs，loveable被替换成lovers;

& 保存搜索字符用来替换其他字符，如s/love/**&**/，love这成**love**;

\< 匹配单词的开始，如:/\
\> 匹配单词的结束，如/love\>/匹配包含以love结尾的单词的行;

x\{m\} 重复字符x，m次，如：/0\{5\}/匹配包含5个0的行;

x\{m,\} 重复字符x，至少m次，如：/0\{5,\}/匹配至少有5个0的行;

x\{m,n\} 重复字符x，至少m次，不多于n次，如：/0\{5,10\}/匹配5~10个0的行;

> 参考：https://www.linuxprobe.com/linux-sed-command.html

调用 shell 文件

PowerShell

```
. 文件名
# 或
source 文件名
```

其他

延时 sleep

PowerShell

```
sleep 1s # 表示延迟一秒
sleep 1m # 表示延迟一分钟
sleep 1h # 表示延迟一小时
sleep 1d # 表示延迟一天
```