

访问统计

283701

『惨绿少年linux』微信公众号



昵称：惨绿少年
园龄：5个月
粉丝：39
关注：26
+加关注

搜索

找找看

谷歌搜索

最新随笔

1. Shell编程基础篇-上
2. Jenkins与网站代码上线解决方案
3. Giti详解及 github与gitlab使用
4. 高并发场景 LVS 安装及高可用实现
5. 企业级Tomcat部署实践及安全调优
6. sed命令详解 vim高级技巧 shell编程上
7. Zabbix 3.0 从入门到精通(zabbix使用详解)
8. inotify+rsync实现实时同步
9. FTP&samba 服务简单部署
10. 用户管理下

Shell编程基础篇-上

1.1 前言

1.1.1 为什么学Shell

Shell脚本语言是实现Linux/UNIX系统管理及自动化运维所必备的重要工具，Linux/UNIX系统的底层及基础应用软件的核心大都涉及Shell脚本的内容。每一个合格的Linux系统管理员或运维工程师，都需要能够熟练地编写Shell脚本语言，并能够阅读系统及各类软件附带的Shell脚本内容。只有这样才能提升运维人员的工作效率，适应日益复杂的工作环境，减少不必要的重复工作，从而为个人的职场发展奠定较好的基础

1.1.2 什么是shell

Shell是一个命令解释器，它在操作系统的最外层，负责直接为用户对话，把用户的输入解释给操作系统，并处理各种各样的操作系统的输出结果，输出屏幕返回给用户。

这种对话方式可以是：

交互的方式：从键盘输入命令，通过/bin/bash的解析，可以立即得到Shell的回应

```
[root@clsn ~]# ls
anaconda-ks.cfg
[root@clsn ~]# echo ls |bash
anaconda-ks.cfg
```

非交互的方式：脚本



1.1.3 什么是Shell脚本

命令、变量和流程控制语句等有机的结合起来

shell脚本擅长处理纯文本类型的数据，而linux中，几乎所有的配置文件，日志，都是纯文本类型文件

1.1.4 脚本语言的种类

一、编译型语言

定义：指用专用的编译器，针对特定的操作平台（操作系统）将某种高级语言源代码一次性翻译成可被硬件平台直接运行的二进制机器码（具有操作数，指令、及相应的格式），这个过程叫做编译（./configure make makeinstall）；编译好的可执行性文件（.exe），可在相对应的平台上运行（移植性差，但运行效率高）。。

典型的编译型语言有，C语言、C++等。

随笔分类(116)		
Linux基础部分(34)		
Linux运维服务部署(36)		
Linux运维网络相关(8)		
Shell编程(1)		
故障解决(6)		
其他(24)		
日常练习题(7)		
积分与排名		
积分 - 86067		
排名 - 3539		
最新评论		
1. Re:Shell编程基础篇-上		
楼主在呀，我来谈谈powershell是怎么实现编码识别的吧。powershell是微软开发的，bom头是微软发明的，powershell脚本自然带有bom头。这个头是用来识别编码的。你在linux中.....		
--PowerShell免费软件		
2. Re:Shell编程基础篇-上		
@PowerShell免费软件👉...		
--惨绿少年		
3. Re:Shell编程基础篇-上		
@Feanmy 还行，还行，哈哈...		
--惨绿少年		
4. Re:Shell编程基础篇-上		
不建议用中文注释？？-----centos 74,一键安装powershell后，就可以用中文注释。中文变量名。		
--PowerShell免费软件		
5. Re:Shell编程基础篇-上		

另外，Java语言是一门很特殊的语言，Java程序需要进行编译步骤，但并不会生成特定平台的二进制机器码，它编译后生成的是一种与平台无关的字节码文件（*.class）（移植性好的原因），这种字节码自然不能被平台直接执行，运行时需要由解释器解释成相应平台的二进制机器码文件；大多数人认为Java是一种编译型语言，但我们说Java即是编译型语言，也是解释型语言也并没有错。

二、解释型语言

定义：指用专门解释器对源程序逐行解释成特定平台的机器码并立即执行的语言；相当于把编译型语言的编译链接过程混到一起同时完成的。

解释型语言执行效率较低，且不能脱离解释器运行，但它的跨平台型比较容易，只需提供特定解释器即可。

常见的解释型语言有，Python（同时是脚本语言）与Ruby等。

三、脚本语言

定义：为了缩短传统的编写-编译-链接-运行（edit-compile-link-run）过程而创建的计算机编程语言。

特点：程序代码即是最终的执行文件，只是这个过程需要解释器的参与，所以说脚本语言与解释型语言有很大的联系。脚本语言通常是被解释执行的，而且程序是文本文件。

典型的脚本语言有，JavaScript，Python，shell等。

其他常用的脚本语句种类

PHP是网页程序，也是脚本语言。是一款更专注于web页面开发（前端展示）的脚本语言，例如：Dedecms,discuz。PHP程序也可以处理系统日志，配置文件等，php也可以调用系统命令。

Perl脚本语言。比shell脚本强大很多，语法灵活、复杂，实现方式很多，不易读，团队协作困难，但仍不失为很好的脚本语言，存世大量的程序软件。MHA高可用Perl写的

Python，不但可以做脚本程序开发，也可以实现web程序以及软件的开发。近两年越来越多的公司都会要求会Python。

Shell脚本与php/perl/python语言的区别和优势？

shell脚本的优势在于处理操作系统底层的业务（linux系统内部的应用都是shell脚本完成）因为有大量的linux系统命令为它做支撑。2000多个命令都是shell脚本编程的有力支撑，特别是grep、awk、sed等。例如：一键软件安装、优化、监控报警脚本，常规的业务应用，shell开发更简单快速，符合运维的简单、易用、高效原则。

PHP、Python优势在于开发运维工具以及web界面的管理工具，web业务的开发等。处理一键软件安装、优化，报警脚本。常规业务的应用等php/python也是能够做到的。但是开发效率和复杂比用shell就差很多了。

系统环境说明

```
[root@clsn scripts]# cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
[root@clsn scripts]# uname -r
3.10.0-693.el7.x86_64
[root@clsn scripts]# getenforce
Disabled
[root@clsn scripts]# systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)
```

1.1.5 系统中的shell

查看系统中的命解释器

```
[root@clsn ~]# cat /etc/shells
/bin/sh
```

写这么多不容易
--Feanmy
阅读排行榜
1. Git详解及 github与gitlab使用(17235)
2. Zabbix 3.0 从入门到精通(zabbix使用详解)(15856)
3. 高并发场景 LVS 安装及高可用实现(12881)
4. inotify+rsync实现实时同步(12717)
5. 企业级Tomcat部署实践及安全调优(12678)
6. 在windows上搭建镜像yum站的方法 (附bat脚本) (11810)
7. linux中的权限(10149)
8. sed命令详解 vim高级技巧 shell编程上(10123)
9. Jenkins与网站代码上线解决方案(9716)
10. linux进程资源占用高原因分析命令记录(8479)

```
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
```



常用操作系统的默认shell

- 1.Linux是Bourne Again shell (bash)
- 2.Solaris和FreeBSD缺省的是Bourne shell (sh)
- 3.AIX下是Korn Shell (ksh)
- 4.HP-UX缺省的是POSIX shell (sh)

```
[root@clsn ~]# echo $SHELL
/bin/bash
```

bash版本



```
[root@clsn scripts]# bash -version
GNU bash, 版本 4.2.46(2)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
许可证 GPLv3+: GNU GPL 许可证版本3或者更高 <http://gnu.org/licenses/gpl.html>
```

这是自由软件，您可以自由地更改和重新发布。
在法律允许的范围内没有担保。



bash 破壳漏洞



```
使用 命令 env x='()' { :;; echo be careful' bash -c "echo this is a test"
如果返回结果为一行，则为正常，
[root@clsn ~]# env x='()' { :;; echo be careful' bash -c "echo this is a test"
this is a test
```

```
#解决办法 升级当前的bash版本
yum install update bash
```



sh与bash 的关系

```
[root@clsn ~]# ll /bin/sh
lrwxrwxrwx. 1 root root 4 11月 13 11:15 /bin/sh -> bash
```

/bin与/user/bin 的关系

```
[root@clsn ~]# ll /bin -d
lrwxrwxrwx. 1 root root 7 11月 13 11:15 /bin -> usr/bin
```

1.2 脚本书写规范

1.2.1 脚本统一存放目录

```
[root@clsn ~]# mkdir -p /server/scripts/
[root@clsn ~]# cd /server/scripts/
```

1.2.2 选择解释器

注意格式 ↓

其中开头的"#!"字符又称为幻数，在执行bash脚本的时候，内核会根据"#!"后的解释器来确定该用那个程序解释这个脚本中的内容。

```
[root@clsn scripts]# head -1 /etc/init.d/*
==> /etc/init.d/functions <==
# -*-Shell-script-*

==> /etc/init.d/netconsole <==
#!/bin/bash

==> /etc/init.d/network <==
#!/bin/bash
```

1.2.3 编辑脚本使用vim

使用 .vimrc 文件，能够快速的生成开头的注释信息

```
[root@clsn scripts]# cat ~/.vimrc
autocmd BufNewFile *.py,*.cc,*.sh,*.java exec ":call SetTitle()"

func SetTitle()
    if expand("%:e") == 'sh'
        call setline(1, "#!/bin/bash")
        call setline(2,
"#####")
        call setline(3, "# File Name: ".expand("%"))
        call setline(4, "# Version: V1.0")
        call setline(5, "# Author: clsn")
        call setline(6, "# Organization: http://blog.znix.top")
        call setline(7, "# Created Time : ".strftime("%F %T"))
        call setline(8, "# Description:")
        call setline(9,
"#####")
        call setline(10, "")
    endif
endfunc
```

使用后的效果

```
[root@clsn scripts]# cat scripts_test.sh
#!/bin/bash
#####
# File Name: scripts_test.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-04 11:39:57
# Description: First scripts file
#####
```

在Shell脚本中，跟在#后面的内容表示注释。注释部分不会被执行，仅给人看。注释可以自成一，也可以跟在命令后面，与命令同行。要养成写注释的习惯，方便自己与他人。

最好不用中文注释，因为在不同字符集的系统会出现乱码。(字符集为zh_CN.UTF-8,为中文)。

1.2.4 文件名规范

名字要有意义，并且结尾以 .sh 结束

1.2.5 开发的规范和习惯小结

- 1) 放在统一的目录
- 2) 脚本以.sh为扩展名
- 3) 开头指定脚本解释器。
- 4) 开头加版本版权等信息，可配置~/.vimrc文件自动添加。
- 5) 脚本不要用中文注释，尽量用英文注释。
- 6) 代码书写优秀习惯
 - a、成对的内容一次性写出来，防止遗漏，如[]、'、'"等
 - b、[]两端要有空格，先输入[]，退格，输入2个空格，再退格写。
 - c、流程控制语句一次书写完，再添加内容。(if 条件 ; then 内容 ; fi)ddd
 - d、通过缩进让代码易读。
 - f、脚本中的引号都是英文状态下的引号，其他字符也是英文状态。

1.3 shell脚本的执行

1.3.1 执行脚本的办法

```
sh/bash scripts.sh
chown +x ./scripts.sh && ./scripts.sh
source scripts.sh
. (空格) scripts.sh
cat oldboyedu.sh |bash # 效率较低
```

source 与 . (点) 的作用

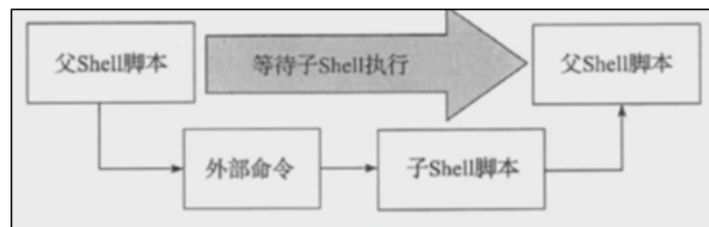
source 命令

```
[root@clsn ~]# help source |head -2
source: source 文件名 [参数]
    在当前 shell 中执行一个文件中的命令。
```

.(点)

```
[root@clsn scripts]# help . |head -2
.: . 文件名 [参数]
    在当前 shell 中执行一个文件中的命令。
```

1.3.2 sh 与 source 的区别



```
[root@clsn scripts]# sh clsn_test.sh
Hello World!
[root@clsn scripts]# echo $clsn
# sh 新建一个Shell窗口 (新建一个进程) 执行一个文件中的命令。

[root@clsn scripts]# source clsn_test.sh
Hello World!
[root@clsn scripts]# echo $clsn
Hello World!
```

面试题:

问sh test.sh后echo \$user返回的结果__空__?

```
[root@oldboy scripts]# cat test.sh
#!/bin/bash
user=`whoami`
```

1.4 Shell的变量

1.4.1 什么是变量

变量可以分为两类：环境变量（全局变量）和普通变量（局部变量）

环境变量也可称为全局变量，可以在创建他们的Shell及其派生出来的任意子进程shell中使用，环境变量又可分为自定义环境变量和Bash内置的环境变量

普通变量也可称为局部变量，只能在创建他们的Shell函数或Shell脚本中使用。普通变量一般是由开发者用户开发脚本程序时创建的。

特殊变量

1.4.2 环境变量

使用 **env/declare/set/export -p** 命令查看系统中的环境变量，这三个命令的输出方式稍有不同。

```
[root@clsn scripts]# env
XDG_SESSION_ID=1
HOSTNAME=clsn
TERM=linux
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=10.0.0.1 5537 22
SSH_TTY=/dev/pts/0
USER=root
~~~
```

输出一个系统中的 环境变量

```
[root@clsn ~]# echo $LANG
zh_CN.UTF-8
```

1.4.3 普通变量

本地变量在用户当前的Shell生存期的脚本中使用。例如，本地变量OLDBOY取值为bingbing，这个值在用户当前Shell生存期中有意义。如果在Shell中启动另一个进程或退出，本地变量值将无效

定义普通变量实践

```
[root@clsn ~]# a=1
[root@clsn ~]# b='2'
[root@clsn ~]# c="3"
[root@clsn ~]# echo "$a"
1
[root@clsn ~]# echo "$b"
2
[root@clsn ~]# echo "${c}"
```

提示：\$变量名表示输出变量，可以用\$c和\${c}两种用法

小结：连续普通字符串内容赋值给变量，不管用什么引号或者不用引号，它的内容是什么，打印变量就输出什么

1.4.4 export命令

```
[root@clsn ~]# help export
export: export [-fn] [名称[=值] ...] 或 export -p
为 shell 变量设定导出属性。
```

标记每个 NAME 名称为自动导出到后续命令执行的环境。如果提供了 VALUE 则导出前将 VALUE 作为赋值。

export命令的说明

当前shell窗口及子shell窗口生效

在新开的shell窗口不会生效，生效需要写入配置文件

定义变量

```
[root@clsn scripts]# CSLN=clsn
[root@clsn scripts]# export CSLN1=1
```

当前窗口查看

```
[root@clsn scripts]# echo $CSLN
clsn
[root@clsn scripts]# echo $CSLN1
1
```

编写测试脚本

```
[root@clsn scripts]# vim quanju.sh
#!/bin/bash
echo $CSLN
echo $CSLN1
```

使用sh执行

```
[root@clsn scripts]# sh quanju.sh

1
```

使用source 执行

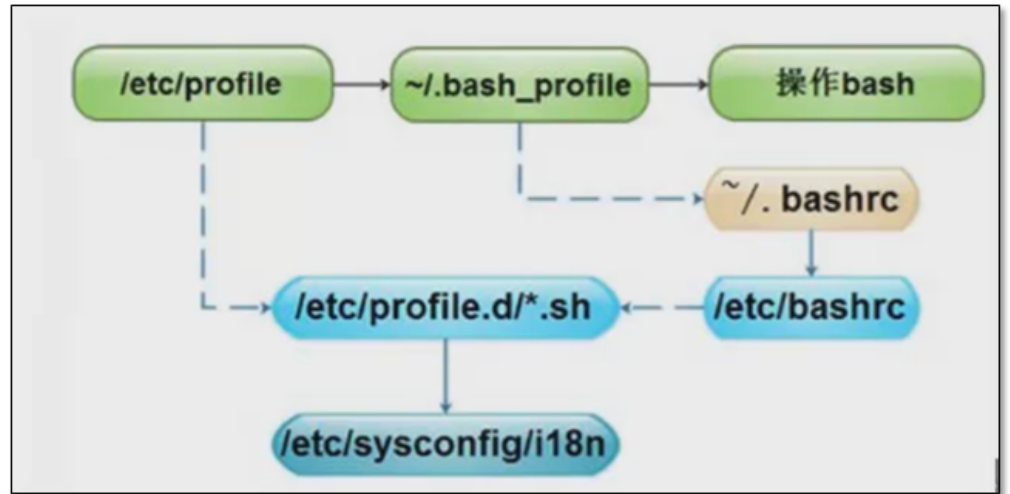
```
[root@clsn scripts]# source quanju.sh
clsn
1
```

1.4.5 环境变量相关配置文件

```
/etc/profile
/etc/bashrc
~/.bashrc
~/.bash_profile
/etc/profile.d/ # 目录
```

四文件读取顺序 (CentOS6和7都一样)

- ① /etc/profile
- ② ~/.bash_profile
- ③ ~/.bashrc
- ④ /etc/bashrc



文件读取过程示意图

验证四文件读取顺序的方法

```

sed -i '1a echo "$(date +%T-%s) /etc/profile1" >>/tmp/clsn' /etc/profile
sed -i '1a echo "$(date +%T-%s) /etc/profile2" >>/tmp/clsn' /etc/profile
sed -i '1a echo "$(date +%T-%s) /etc/bashrc1" >>/tmp/clsn' /etc/bashrc
sed -i '1a echo "$(date +%T-%s) /etc/bashrc2" >>/tmp/clsn' /etc/bashrc
sed -i '1a echo "$(date +%T-%s) ~/.bashrc1" >>/tmp/clsn' ~/.bashrc
sed -i '1a echo "$(date +%T-%s) ~/.bashrc2" >>/tmp/clsn' ~/.bashrc
sed -i '1a echo "$(date +%T-%s) ~/.bash_profile1" >>/tmp/clsn' ~/.bash_profile
sed -i '1a echo "$(date +%T-%s) ~/.bash_profile2" >>/tmp/clsn' ~/.bash_profile

```

1.4.6 环境变量的知识小结

- ❏ 变量名通常要大写。
- ❏ 变量可以在自身的Shell及子Shell中使用。
- ❏ 常用export来定义环境变量。
- ❏ 执行env默认可以显示所有的环境变量名称及对应的值。
- ❏ 输出时用"\$变量名"，取消时用"unset 变量名"。
- ❏ 书写crond定时任务时要注意，脚本要用到的环境变量最好先在所执行的Shell脚本中重新定义。
- ❏ 如果希望环境变量永久生效，则可以将其放在用户环境变量文件或全局环境变量文件里。

1.4.7 变量中引号的使用

只有在变量的值中有空格的时候，会使用引号。

单引号与双引号的区别在于，是否能够解析特殊符号。

```

[root@clsn ~]# name=znix
[root@clsn ~]# name2='clsn'
[root@clsn ~]# name3="http://blog.znix.top"
[root@clsn ~]# echo $name
znix
[root@clsn ~]# echo $name2
clsn
[root@clsn ~]# echo $name3
http://blog.znix.top
[root@clsn ~]# name4='cl sn'
[root@clsn ~]# echo $name4
cl sn
[root@clsn ~]# name5="cl sn"
[root@clsn ~]# echo $name5
cl sn

```



```

cl sn
[root@clsn ~]# name6='cl sn $PWD'
[root@clsn ~]# echo $name6
cl sn $PWD
[root@clsn ~]# name6="cl sn $PWD"
[root@clsn ~]# echo $name6
cl sn /root

```

1.4.8 普通变量的要求

- 1) 内容是纯数字、简单的连续字符（内容中不带任何空格）时，定义时可以不加任何引号，例如：

```

a.ClsnAge=22
b.NETWORKING=yes

```

- 2) 没有特殊情况时，字符串一律用双引号定义赋值，特别是多个字符串中间有空格时，例如：

```

a.NFSD_MODULE="no load"
b.MyName="Oldboy is a handsome boy."

```

- 3) 当变量里的内容需要原样输出时，要用单引号（M），这样的需求极少，例如：

```

a.OLDBOY_NAME='OLDBOY'

```

变量使用反引号赋值

```

[root@clsn scripts]# time=`date`
[root@clsn scripts]# echo $time
2017年 12月 05日 星期二 09:02:06 CST

[root@clsn scripts]# file=`ls`
[root@clsn scripts]# echo $file
clsn_test.sh panduan.sh quanju.sh yhk.sh

```

使用\${}

打印变量的时候防止出现“金庸新著”的问题

```

[root@clsn scripts]# time=`date`
[root@clsn scripts]# echo $time_day

[root@clsn scripts]# echo ${time}_day
2017年 12月 05日 星期二 09:02:06 CST_day
[root@clsn scripts]# echo $time-day
2017年 12月 05日 星期二 09:02:06 CST-day

```

编写脚本测试\${}

```

# 使用脚本测试
[root@clsn scripts]# vim bianliang.sh
#!/bin/bash
#####
# File Name: bianliang.sh
# Version: V1.0
# Author: clsn

```

```
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:10:29
# Description:
#####

time=`date`
echo $timeday
echo ${time}day

[root@clsn scripts]# sh bianliang.sh

2017年 12月 05日 星期二 09:11:19 CSTday
```

1.4.9 定义变量名技巧

- 1. 变量名只能为字母、数字或下划线，只能以字母或下划线开头。
- 2. 变量名的定义要有一定的规范，并且要见名知意。

示例:

```
ClsnAge=22      #<==每个单词的首字母大写的写法

cls_n_age=22    #<==单词之间用"_"的写法

cls_nAgeSex=man #<==驼峰语法：首个单词的首字母小写，其余单词首字母大写

CLS_NAGE=22     #<==单词全大写的写法
```

- 3. 一般的变量定义、赋值常用双引号；简单连续的字符串可以不加引号；希望原样输出时使用单引号。
- 4. 希望变量的内容是命令的解析结果时，要用反引号"，或者用\$()把命令括起来再赋值。

1.5 特殊变量

1.5.1 位置变量

常用的特殊位置参数说明

位置变量	作用说明
\$0	获取当前执行的shell脚本的文件名，如果执行脚本带路径那么就包括脚本路径。
\$n	获取当前执行的shell脚本的第n个参数值，n=1..9，当n为0时表示脚本的文件名，如果n大于9用大括号括起来{10}，参数以空格隔开。
\$#	获取当前执行的shell脚本后面接的参数的总个数
\$*	获取当前shell的所有传参的参数，不加引号同\$@;如果给\$*加上双引号，例如：“\$*”，则表示将所有的参数视为单个字符串，相当于“12\$3”。
\$@	获取当前shell的所有传参的参数，不加引号同\$*;如果给\$@加上双引号，例如：“\$@”，则表示将所有参数视为不同的独立字符串，相当于“\$1” “\$2” “\$3” “.....”，这是将参数传递给其他程序的最佳方式，因为他会保留所有内嵌在每个参数里的任何空白。
当“\$*”和“\$@”都加双引号时，两者有区别，都不加双引号时，两者无区别。	

0,1,\$2 ~ 参数实践

```
[root@clsn scripts]# vim chanshu.sh
#!/bin/bash
#####
# File Name: chanshu.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:39:16
```

```
# Description:
#####

echo $0
echo "第一个参数: " $1
echo "第二个参数: " $2
echo "第11个参数: " ${11}
[root@clsn scripts]# sh chanshu.sh
chanshu.sh
第一个参数:
第二个参数:
第11个参数:
[root@clsn scripts]# sh chanshu.sh 1 2 3 4 5 6 7 8 9 10 11
chanshu.sh
第一个参数: 1
第二个参数: 2
第11个参数: 11
```

\$# 参数实践

```
[root@clsn scripts]# vim chanshu.sh
#####
# File Name: chanshu.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:39:16
# Description:
#####

echo $0
echo "第一个参数: " $1
echo "第二个参数: " $2
echo "第10个参数: " ${10}
echo "第11个参数: " ${11}
echo "参数个数: " $#

[root@clsn scripts]# sh chanshu.sh 55 2 3 4 5 6 7 8 9 10 11 112
chanshu.sh
第一个参数: 55
第二个参数: 2
第10个参数: 10
第11个参数: 11
参数个数: 12
```

\$* 参数实践

```
[root@clsn scripts]# vim chanshu.sh
#####
# File Name: chanshu.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:39:16
# Description:
#####

echo $0
echo "第一个参数: " $1
echo "第二个参数: " $2
```

```
echo "第10个参数: " ${10}
echo "第11个参数: " ${11}
echo "参数个数: " $#
echo "参数:" $*
"chanshu.sh" 18L, 456C 已写入
[root@clsn scripts]# sh chanshu.sh 55 2 3 4 5 6 7 8 9 10 11 112
chanshu.sh
第一个参数: 55
第二个参数: 2
第10个参数: 10
第11个参数: 11
参数个数: 12
参数: 55 2 3 4 5 6 7 8 9 10 11 112
```

\$* 与 @\$ 对比实践

```
[root@clsn scripts]# set -- "I am" handsome boy..
[root@clsn scripts]# echo $1
I am
[root@clsn scripts]# echo $2
handsome
[root@clsn scripts]# echo $3
boy..
[root@clsn scripts]# echo $*
I am handsome boy..
[root@clsn scripts]# echo @$
I am handsome boy..

[root@clsn scripts]# for i in $*;do echo $i ;done
I
am
handsome
boy..
[root@clsn scripts]# for i in $@;do echo $i ;done
I
am
handsome
boy..
[root@clsn scripts]# for i in "$@";do echo $i ;done
I am
handsome
boy..
[root@clsn scripts]# for i in "$*";do echo $i ;done
I am handsome boy..
```

1.5.2 进程状态变量

Shell进程的特殊状态变量说明

位置变量	作用说明
\$?	获取执行上一个指令的执行状态返回值（0为成功，非零为失败），这个变量最常用
\$\$	获取当前执行的Shell脚本的进程号（PID），这个变量不常用，了解即可
#!	获取上一个在后台工作的进程的进程号（PID），这个变量不常用，了解即可
\$_	获取在此之前执行的命令或脚本的最后一个参数，这个变量不常用，了解即可

进程参数实践

```
[root@clsn scripts]# echo $?
0
[root@clsn scripts]# echo $$
```

```
1368
[root@clsn scripts]# echo $!

[root@clsn scripts]# echo $_
echo
```

1.5.3 echo参数说明

参数	参数说明
-n	不要追加换行
-e	启用下列反斜杠转义的解释
-E	显式地抑制对于反斜杠转义的解释
`echo' 对下列反斜杠字符进行转义:	
\n	换行
\r	回车
\t	横向制表符
\b	退格
\v	纵向制表符
\c	抑制更多的输出

1.6 定义变量的方式

1.6.1 三种定义变量的方式

- 1、直接赋值
- 2、传参 (传递参数)
- 3、交互式设置变量，使用read命令

1.6.2 read命令说明

在命令行中使用

```
[root@clsn scripts]# read
132
[root@clsn scripts]# echo $REPLY
132
[root@clsn scripts]# read clsn
456
[root@clsn scripts]# echo $clsn
456
[root@clsn scripts]# echo $REPLY
132
```

在脚本中使用

```
[root@clsn scripts]# vim clsn_test.sh
#!/bin/bash
read -p '请输入:' clsn

echo $clsn
```


执行结果

```
[root@clsn scripts]# sh clsn_test.sh
请输入:clsn_znix
clsn_znix
```

read命令的帮助说明



```
[root@clsn scripts]# read --help
-bash: read: --: 无效选项
read: 用法: read [-ers] [-a 数组] [-d 分隔符] [-i 缓冲区文字] [-n 读取字符数] [-N 读取字符数]
[-p 提示符] [-t 超时] [-u 文件描述符] [-s 不显示终端的任何输入] [名称 ...]
```

1.6.3 定义方法实践

直接赋值方法




```
[root@clsn scripts]# vim bianliang.sh
# File Name: bianliang.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:10:29
# Description:
#####
name=CLSN
age=22
sex=Man
hobby=`ls`
ethFile=/etc/sysconfig/network-scripts/ifcfg-eth0

echo $hobby
ls $ethFile
[root@clsn scripts]# sh bianliang.sh
bianliang.sh chanshu.sh clsn.sh clsn_test.sh panduan.sh quanju.sh xiugaizhuji.sh
yhk.sh
/etc/sysconfig/network-scripts/ifcfg-eth0
```


传参(传递参数)


```
[root@clsn scripts]# vim bianliang.sh
#####
# File Name: bianliang.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 09:10:29
# Description:
#####
name=CLSN
age=22
sex=Man
hobby=$1
ethFile=$2

echo $hobby
ls $ethFile
[root@clsn scripts]# sh bianliang.sh clsn /etc/hostname
clsn
/etc/hostname
```


交互式设置变量 read


```
[root@clsn scripts]# vim yhk.sh
#!/bin/bash
#####
# File Name: yhk.sh
# Version: V1.0
```

```
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-04 17:01:44
# Description:
#####
read -p "请输入你的银行卡号:" Yhk
read -s -p "请输入密码:" miMa
echo
echo "你的银行卡号:" $Yhk
echo "你的密码为:" $miMa
# 测试结果
[root@clsn scripts]# sh yhk.sh
请输入你的银行卡号: 123456
请输入密码:
你的银行卡号: 123456
你的密码为: 123456
```

1.6.4 写一个交互脚本，实现能够定义主机名及IP地址

脚本内容↓

```
[root@clsn scripts]# cat xiugaizhuji.sh
#!/bin/bash
#####
# File Name: jiaohu.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 10:55:42
# Description:
#####

ethFile=/etc/sysconfig/network-scripts/ifcfg-eth[01]
Now_eth=`hostname -I|awk -F "." '{print $4}'`

read -p "请输入主机名:" Hostname
read -p "请输入IP地址的主机位:" HostIP

hostnamectl set-hostname $Hostname

sed -i "s#${Now_eth}#${HostIP}#g" $ethFile

read -p "是否重启服务器:{yes/no}" REboot

if [ $REboot == yes ]
then
    echo "系统将在10秒后重启!"
    shutdown -r 10
else
    echo "请稍后手动重启系统!"
fi
```

脚本测试结果

```
[root@clsn scripts]# sh xiugaizhuji.sh
请输入主机名:clsn
请输入IP地址的主机位:180
是否重启服务器:{yes/no}yes
系统将在10秒后重启!
[root@clsn scripts]# sh xiugaizhuji.sh
请输入主机名:clsn
请输入IP地址的主机位:180
```

是否重启服务器: {yes/no}no
请稍后手动重启！



1.7 变量的子串

1.7.1 变量子串说明

表达式	说明
<code>\${parameter}</code>	返回变量 <code>\$parameter</code> 的内容
<code>\${#parameter}</code>	返回变内容的长度（按字符），也适用于特殊变量
<code>\${parameter:offset}</code>	在变量 <code>\${parameter}</code> 中，从位置 <code>offset</code> 之后开始提取子串到结尾
<code>\${parameter:offset:length}</code>	在变量 <code>\${parameter}</code> 中，从位置 <code>offset</code> 之后开始提取长度为 <code>length</code> 的子串
<code>\${parameter#word}</code>	从变量 <code>\${parameter}</code> 开头开始删除最短匹配的 <code>word</code> 子串
<code>\${parameter##word}</code>	从变量 <code>\${parameter}</code> 开头开始删除最长匹配的 <code>word</code> 子串
<code>\${parameter%word}</code>	从变量 <code>\${parameter}</code> 结尾开始删除最短匹配的 <code>word</code> 子串
<code>\${parameter%%word}</code>	从变量 <code>\${parameter}</code> 结尾开始删除最长匹配的 <code>word</code> 子串
<code>\${parameter/pattem/string}</code>	使用 <code>string</code> 代替第一个匹配的 <code>pattern</code>
<code>\${parameter//pattem/string}</code>	使用 <code>string</code> 代替所有匹配的 <code>pattern</code>

计算变赋值的长度



```
[root@clsn scripts]# clsn=http://blog.znix.top
[root@clsn scripts]# echo ${clsn} |wc -L
20
[root@clsn scripts]# echo ${#clsn}
20
[root@clsn scripts]# time echo ${clsn} |wc -L
20

real    0m0.002s
user    0m0.002s
sys     0m0.000s
[root@clsn scripts]# time echo ${#clsn}
20

real    0m0.000s
user    0m0.000s
sys     0m0.000s
```



截取变量中的字符



```
[root@clsn scripts]# clsn=abcABC123ABCabc
[root@clsn scripts]# echo ${clsn#abc}
ABC123ABCabc
[root@clsn scripts]# echo ${clsn##abc}
ABC123ABCabc
[root@clsn scripts]# echo ${clsn%abc}
abcABC123ABC
[root@clsn scripts]# echo ${clsn%%abc}
abcABC123ABC
[root@clsn scripts]# echo ${clsn#a*c}
ABC123ABCabc
[root@clsn scripts]# echo ${clsn##a*c}
```



```
[root@clsn scripts]# echo ${clsn%a*c}
abcABC123ABC
[root@clsn scripts]# echo ${clsn%%a*c}

[root@clsn scripts]# echo ${clsn#a*C}
123ABCabc
[root@clsn scripts]# echo ${clsn#*a*C}
123ABCabc
[root@clsn scripts]# echo ${clsn##a*C}
abc
[root@clsn scripts]# echo ${clsn%a*c}
abcABC123ABC
[root@clsn scripts]# echo ${clsn%A*c}
abcABC123
[root@clsn scripts]# echo ${clsn%%A*c}
abc
```

替换变量内容

```
[root@clsn scripts]# echo $clsn
abcABC123ABCabc
[root@clsn scripts]# echo ${clsn/abc/clsn}
clsnABC123ABCabc
[root@clsn scripts]# echo ${clsn//abc/clsn}
clsnABC123ABCclsn
```

有关上述匹配删除的小结

#表示从开头删除匹配最短。
##表示从开头删除匹配最长。
%表示从结尾删除匹配最短。
%%表示从结尾删除匹配最长。
a*c表示匹配的字符串，*表示匹配所有，a*c匹配开头为a、中间为任意多个字符、结尾为c的字符串。
a*C表示匹配的字符串，*表示匹配所有，a*C匹配开头为a、中间为任意多个字符、结尾为C的字符串。

有关替换的小结


一个“/”表示替换匹配的第-个字符串。
两个“/”表示替换匹配的所有字符串。

1.7.2 Shell的特殊扩展变量说明


表达式	说明
<code>\${parameter:-word}</code>	如果parameter的变量值为空或未赋值，则会返回word字符串并替代变量的值用途.如果变量未定义，则返回备用的值，防止变量为空值或因未定义而导致异常
<code>\${parameter:=word}</code>	如果parameter的变量值为空或未赋值，则设置这个变量值为word,并返回其值。位置变量和特殊变量不适用用途：基本同上一个 <code>\${parameter>word}</code> ，但该变量又额外给parameter变量赋值了
<code>\${parameter:?word}</code>	如果parameter变量值为空或未赋值，那么word字符串将被作为标准错误输出，否则输出变量的值。用途：用于捕捉由于变量未定义而导致的错误，并退出程序
<code>\${parameter:+word}</code>	如果parameter变量值为空或未赋值，则什么都不做，否则word字符串将替代变量的值

特殊变量实践


脚本内容



```
[root@clsn scripts]# cat clsn.sh
#!/bin/bash
#####
# File Name: clsn.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-05 12:13:38
# Description:
#####
dir=
echo ${dir:-/tmp}
echo ${dir}
echo ${dir:=/mnt}
echo ${dir}
dir2= (空格)
echo ${dir2:-/tmp}
echo ${dir2}
echo ${dir2:-/tmp}
echo ${dir2}
echo ${dir2:=/mnt}
echo ${dir2}
```




测试结果



```
[root@clsn scripts]# sh clsn.sh
/tmp

/mnt
/mnt

/tmp
```




至此shell中的变量就都介绍完了


1.8 变量的数值计算

1.8.1 仅支持整数的运算

echo \$((数学运算表达式))



```
# 形式一
[root@clsn scripts]# echo $((1 + 1))
2
[root@clsn scripts]# echo $((2*7-3/6+5))
19
# 形式二
[root@clsn scripts]# ((clsn=2*8))
[root@clsn scripts]# echo $clsn
16
# 形式三
[root@clsn scripts]# znix=$((2*7-3/6+5))
[root@clsn scripts]# echo $znix
19
```



延伸产物(重要)

i++ 自增1

```
i-- 自减1  
  
++i  
  
--i
```

示例:

```
[root@clsn scripts]# i=1  
[root@clsn scripts]# echo $((i++))  
1  
[root@clsn scripts]# echo $((i++))  
2  
[root@clsn scripts]# echo $((i--))  
3  
[root@clsn scripts]# echo $((i--))  
2  
[root@clsn scripts]# echo $((i--))  
1  
[root@clsn scripts]# echo $((++i))  
1  
[root@clsn scripts]# echo $((++i))  
2  
[root@clsn scripts]# echo $((++i))  
3  
[root@clsn scripts]# echo $((--i))  
2  
[root@clsn scripts]# echo $((--i))  
1  
[root@clsn scripts]# echo $((--i))  
0
```

记忆方法: ++, --

变量a在前, 表达式的值为a, 然后a自增或自减, 变量a在符号后, 表达式值自增或自减, 然后a值自增或自减。

let命令

```
[root@clsn scripts]# i=1  
[root@clsn scripts]# i=i+1  
[root@clsn scripts]# echo $i  
i+1  
  
[root@clsn scripts]# i=1  
[root@clsn scripts]# let i=i+1  
[root@clsn scripts]# echo $i  
2
```

expr 命令

1. 整数计算
2. 判断扩展名
3. 判断输入是否为整数, 非整数返回值为2
4. 计算变量的长度

```
[root@clsn scripts]# expr 1+1  
1+1  
[root@clsn scripts]# expr 1 + 1
```

```
2
[root@clsn scripts]# expr 1 * 1
expr: 语法错误
[root@clsn scripts]# expr 1 \* 1
1
```

非整数返回值为2 示例:

```
[root@clsn scripts]# expr 1 + 1
2
[root@clsn scripts]# echo $?
0
[root@clsn scripts]# expr -1 + 1
0
[root@clsn scripts]# echo $?
1
[root@clsn scripts]# expr a + 1
expr: 非整数参数
[root@clsn scripts]# echo $?
2
```

\$[]运算符

```
[root@clsn scripts]# echo ${1+2}
3
[root@clsn scripts]# echo ${1-2}
-1
[root@clsn scripts]# echo ${1*2}
2
[root@clsn scripts]# echo ${1/2}
0
```

typeset命令进行运算

```
[root@clsn scripts]# typeset -i A=2017 B=2018
[root@clsn scripts]# A=A+B
[root@clsn scripts]# echo $A
4035
```

1.8.2 可以进行小数运算的命令

bc 命令

```
# 安装 bc 依赖于base源
[root@clsn scripts]# yum -y install bc
```

交互模式测试bc命令

```
[root@clsn scripts]# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
1+1
2
[root@clsn scripts]# echo 1+1.1|bc
2.1
```

免交互模式测试bc命令

```
[root@clsn scripts]# echo 'scale=6;1/3'|bc
.333333
```

python 命令

```
[root@clsn scripts]# file `which yum`
/usr/bin/yum: Python script, ASCII text executable
[root@clsn scripts]# python
>>> import os
>>> os.system('df -h')
>>> 1+1.1
2.1
>>>exit()
```

awk 命令

```
[root@clsn ~]# echo "7.7 3.8"|awk '{print ($1-$2)}'
3.9
[root@clsn ~]# echo "358 113"|awk '{print ($1-3)/$2}'
3.14159
[root@clsn ~]# echo "3 9"|awk '{print ($1+3)*$2}'
54
[root@backup scripts]# awk BEGIN'{print 1.2+3.3}'
4.5
```

1.8.3 运算相关练习题

1.8.3.1 【练习题】实现一个加减乘除等功能的计算器

实现脚本:

```
[root@clsn scripts]# cat jishuanqi.sh
#!/bin/bash
#####
# File Name: jishuanqi.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-06 08:57:13
# Description:
#####

read -p "请输入第一个整数:" a
read -p "请输入第二个整数:" b

echo $a + $b=$((a+$b))
echo $a - $b=$((a-$b))
echo $a \* $b=$((a*$b))
echo $a / $b=$((a/$b))
```

脚本执行过程:

```
[root@clsn scripts]# sh jishuanqi.sh
请输入第一个整数:12
请输入第二个整数:12
12 + 12 =24
12 - 12 =0
```

```
12 * 12 =144
12 / 12 =1
```



精简方法



```
[root@clsn scripts]# vim jishuanqi2.sh
#!/bin/bash
#####
# File Name: jishuanqi2.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-06 15:02:41
# Description:
#####
echo $(( $1 ))
```



脚本执行过程:

```
[root@clsn scripts]# sh jishuanqi2.sh 1+1
2
[root@clsn scripts]# sh jishuanqi2.sh 1*9
9
```

1.8.3.2 【练习题】打印结果1+2+3+4+5+6+7+8+9+10=55

脚本内容



```
[root@clsn scripts]# vim yunshuan.sh
#!/bin/bash
#####
# File Name: yunshuan.sh
# Version: V1.0
# Author: clsn
# Organization: http://blog.znix.top
# Created Time : 2017-12-06 09:40:31
# Description:
#####

Num=`seq -s + 1 10`
echo $Num=$(( $Num ))
```



脚本执行结果

```
[root@clsn scripts]# sh yunshuan.sh
1+2+3+4+5+6+7+8+9+10=55
```

1.9 补充说明

shell脚本中批量注释的方法

```
<<'EOF'
文件内容
EOF
或
使用 exit可以注释其之后的所有内容(类似注释,实质为不执行后面的内容)
```

1.9.1 参考文献

<http://blog.csdn.net/lansesi2008/article/details/20558369>

https://www.abcdocker.com/abcdocker/269
http://blog.51cto.com/life2death/1657133

本文出自“惨绿少年”，欢迎转载，转载请注明出处！http://blog.znix.top

本博文中2017年11月11日之前使用的系统版本为: CentOS release 6.9 (Final) 内核版本为: 2.6.32-696.10.1.el6.x86_64
在2017年11月11日之后发布的博文使用的系统版本为: CentOS Linux release 7.4.1708 (Core) 内核版本为: 3.10.0-693.el7.x86_64

大家如果有什么问题可以留言或邮件联系 admin@znix.top，我看到后会尽快回复。
本文出自“惨绿少年”博客，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出。
来源:<http://blog.znix.top>

分类: [Linux基础部分](#), [Shell编程](#), [日常练习题](#)

好文要顶 关注我 收藏该文





惨绿少年
关注 - 26
粉丝 - 39
[+加关注](#)

3 0

« 上一篇: [Jenkins与网站代码上线解决方案](#)

posted @ 2017-12-06 16:01 惨绿少年 阅读(1766) 评论(5) 编辑 收藏

评论列表

- #1楼 2017-12-06 16:12 Feanmy
写这么多不容易
支持(0) 反对(0)
- #2楼 2017-12-06 16:48 PowerShell免费软件
不建议用中文注释？？-----centos 74,一键安装powershell后，就可以用中文注释。中文变量名。
支持(0) 反对(0)
- #3楼[楼主] 2017-12-06 16:48 惨绿少年
@ Feanmy 还行，还行，哈哈
支持(0) 反对(0)
- #4楼[楼主] 2017-12-06 16:58 惨绿少年
@ PowerShell免费软件
支持(0) 反对(0)
- #5楼 2017-12-06 17:30 PowerShell免费软件
楼主在呀，我来谈谈powershell是怎么实现编码识别的吧。
powershell是微软开发的，bom头是微软发明的，powershell脚本自然带有bom头。这个头是用来识别编码的。

你在linux中生成一个a.ps1

```
pwsh -command "add-content /tmp/a.ps1 -value '$哈 = 1' -Encoding Unicode"
```

就产生了个bom头+utf16编码的文件。

在zh-cn.utf8的环境下，用vi打开这个文件，不会乱码。因为vi支持bom头。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】腾讯云免费实验室，1小时搭建人工智能应用

【新闻】H3 BPM体验平台全面上线



最新IT新闻:

- 量子基金创始人：20世纪属于美国，但21世纪属于中国
 - 亚马逊：我不卖你家的东西，Google：那YouTube你也别想看
 - 马云乌镇6000字演讲全文：转型很痛苦但值得，相信“相信”的力量
 - 2.3亿中国人手机种树震撼老外：蚂蚁森林走红联合国
 - 阿里第一次将售假卖家告到了互联网法院
- » 更多新闻...



最新知识库文章:

- 以操作系统的角度述说线程与进程
 - 软件测试转型之路
 - 门内门外看招聘
 - 大道至简，职场上做人做事做管理
 - 关于编程，你的练习是不是有效的？
- » 更多知识库文章...

Copyright ©2017 惨绿少年

本作品采用知识共享署名-非商业性使用 3.0 未本地化版本许可协议进行许可。

[TOP](#)