

Bash 脚本编程基础 - v7

samli@tencent.com

2010-08

课程目标

- 掌握 Bash 的类型与配置
- 掌握 Bash 基本语法
- 掌握监控程序的编写
- 掌握 Bash 模块化编程

内容导航

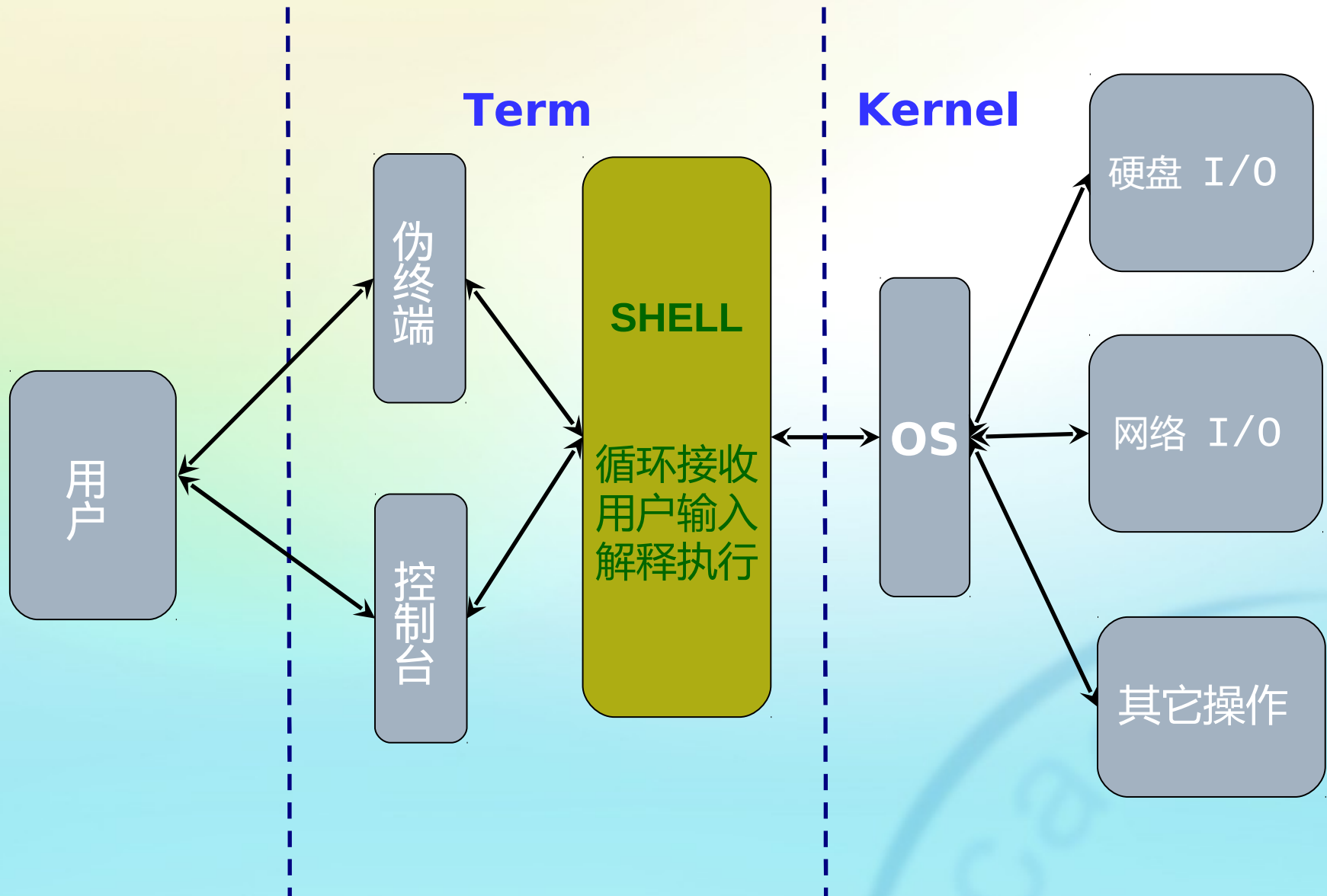
- Bash 基础
- Bash 语法
- 监控实例



Bash 基础

- Shell 与 Bash
- Bash 的类型
- Bash 命令的解释过程
- Bash 与 Vim 配置

Shell



Bash 的类型

Login Bash

Interactive Bash

Bash Scripts

Login Bash

- 什么是 Login Bash
- Login 时 Bash 如何初始化

全局配置

/etc/profile

个人配置

~/.bash_profile

~/.bash_login

~/.profile

这个又是啥？

/etc/profile.d/

SUSE： 确保文件可读

SLK： 确保文件可读可执行

Login sh

- login sh 和 login bash 的初始化有什么不同

Login sh 只读取：

- /etc/profile
- ~/.profile

- 查看：ps; /etc/passwd

- 指定：useradd -s /bin/bash

- 修改：usermod, chsh

Interactive Bash

- 什么是交互式 Bash
- 启动交互式 Bash 时，Bash 如何初始化

`/etc/bash.bashrc`

`~/.bashrc`

source 命令

- Login bash 与 interactive bash 的统一初始化
在 login 的时候自动执行 ~/.bashrc

```
if [ -f "$HOME/.bashrc" ]; then  
    . "$HOME/.bashrc"  
fi
```

更详细资料见 man bash: [INVOCATION]

Bash 脚本

常见问题：

- `#! /bin/bash` 的位置
- 来自 windows 的 CRLF 的换行符
- 命令的结束符（分割符）
- 脚本的权限位
- 文件掩码（`umask`）
- 脚本的退出与退出码

Bash 脚本的执行

- 父 shell Fork 一 sub-shell
- Sub-shell 继承父 shell 环境
- Sub-shell 调用指定的解释器解释执行该脚本
- Sub-shell 退出，相关环境被销毁
- 父 shell 取得 sub-shell 的退出状态

Bash 命令的解析

- 1、brace expansion (花括号扩展)
- 2、tilde expansion (波浪线扩展)
- 3、parameter,variable expansion (变量扩展)
- 4、arithmetic expansion (算术扩展)
- 5、command substitution (命令替换)
- 6、word splitting (词的拆分)
- 7、pathname expansion (路径名扩展)

Bash 脚本与 Vim 配置

- `/etc/vimrc` 或 `~/.vimrc` 中配置

<code>syntax on</code>	" 语法高亮
<code>set number</code>	" 显示行数
<code>set shiftwidth=4</code>	" 四格缩进
<code>set tabstop=4</code>	" 四格缩进
<code>set expandtab</code>	" tab 转 space
<code>set bg=light</code>	" 背景为亮色
<code>set paste</code>	" 进入粘贴模式
<code>color murphy</code>	" 配色方案

内容导航

- Bash 基础
- Bash 语法
- 监控实例

√



Bash 语法

- 变量
- 数值运算
- 流程控制
- 重定向
- 函数

变量

- 变量赋值：`name=value`
- 变量引用：`$name`
- 合法的变量名：字母、下划线、数字

注意：

- `=` 号两边不允许空格
- `$name` 是 `${name}` 的简写
- 需要使用 `${name}` 的两种情况

赋值与扩展

- 1、brace expansion (花括号扩展)
- 2、tilde expansion (波浪线扩展)
- 3、parameter,variable expansion (变量扩展)
- 4、arithmetic expansion (算术扩展)
- 5、command substitution (命令替换)
- 6、word splitting (词的拆分)
- 7、pathname expansion (路径名扩展)

见 man bash 之 [PARAMETERS]

赋值与命令替换

- 命令替换

- `` command ``
- `$(command)` # 推荐

例

- `files=$(ls *.bak)`
- `today=$(date +%F)`
- `lines=$(cat file)`

常用变量

- \$? 上个命令的退出状态
- \$! 最后一个后台进程的 pid
- \$0 当前脚本的名字
- \$\$ 当前脚本的 pid
- \$n n 为 1, 2...n, 脚本或函数的参数
- \$@ 脚本或函数的所有参数
- \$# 脚本或函数的参数个数

常用变量

• \$UID	当前用户的 uid
• \$LOGNAME	当前用户名
• \$HOSTNAME	主机名
• \$RANDOM	1-32767 间的随机数
• \$SECONDS	已经消耗的秒数
• \$PWD	当前的工作目录
• \$OLDPWD	最后一次 cd 前的工作目录
• \$HISTTIMEFORMAT	?

环境变量

- 环境变量是 Bash 或用户预设置的变量，可被继承并直接使用
- 有些环境变量会影响 shell 的行为

• \$PATH	命令搜索路径名
• \$LANG	locale
• \$LC_ALL	locale
• \$TERM	终端类型，一般为 xterm
• \$EDITOR	系统默认编辑器

环境变量的设置

- 永久设置

```
export name=value  
bash foofoo.sh
```

- 只对一个命令设置

```
name=value ./foofoo.sh
```

- 更多信息请 `man env`
- 更多变量见 `man bash: [Shell Variables]`

问题

- 在哪设置 JAVA_HOME , 让所有用户皆可使用
- 按上面的设置后 , rc.local 中 执行以下命令可否成功
 - /usr/local/apache-tomcat/bin/start.sh

环境变量要注意的

- 变量需要 `export` 才能被子进程看到
 - 成为环境变量
- 大部分的环境变量在 `login` 时被设置
 - 而不是在 OS 启动时
- 要注意 `locale` 对程序的影响
 - 建议在脚本开头统一设置

数值运算

- 运算工具
 - let, expr
 - (())
 - bc

- 常用运算
 - + - * / %
(加, 减, 乘, 除, 取模)

```
i=10; j=20;  
(( i++ ))  
(( cnt = i + j ))  
cnt=$(( i + j ))  
(( k = i * j ))  
(( k = j / i ))  
(( k = j % i ))
```

更详细内容：man bash: [ARITHMETIC EVALUATION]

管道与 here doc

bc 支持浮点数运算

但 bc 是交互式的，怎么办？

```
echo "scale = 3; 10 / 3" | bc
```

```
bc <<EOF          # EOF 称为占位符
scale = 3          # 输入的内容
10 / 3
EOF                # 本行不能有其它字符
```

流程控制

- 条件判断
 - if
 - && 与 ||
 - case
- 循环
 - for, while
 - break, continue

if ... then

- `if ...` 语句用于判断一个 / 一组命令是执行否成功
- `[[...]]` 表达式, 测试字符串, 文件
- `((...))` 表达式, 测试数值计算结果
- `bash` 命令, 测试命令是否执行成功
- 命令的退出码 (`$?`) 为 0 时, 代表 `true`

字符串测试

测试	意义
<code>[[-n \$str]]</code>	<code>\$str</code> 不为空值
<code>[[-z \$str]]</code>	<code>\$str</code> 为空值
<code>[[\$str1 == \$str2]]</code>	相等
<code>[[\$str1 != \$str2]]</code>	不等

文件测试

用法	意义	用法	意义
<code>[[-b \$file]]</code>	块设备	<code>[[-r \$file]]</code>	文件可读
<code>[[-c \$file]]</code>	字符设备	<code>[[-s \$file]]</code>	文件大小不为零
<code>[[-d \$file]]</code>	目录	<code>[[-w \$file]]</code>	文件可写
<code>[[-e \$file]]</code>	文件存在	<code>[[-x \$file]]</code>	文件可执行
<code>[[-f \$file]]</code>	普通文件	<code>[[-p \$file]]</code>	命名管道

- 更详细内容 `man bash [CONDITIONAL EXPRESSIONS]`

if ... else ...

判断 OS

```
if [[ -f /etc/SuSE-release ]]; then
    echo SUSE OS

elif [[ -f /etc/slackware-version ]]; then
    echo Slackware OS

else
    echo "Redhat ?"
fi
```


命令的退出状态

- 检查 \$? 的值是否为 0, 0 为成功, 非 0 为失败

```
grep -q "^admin:" /etc/passwd
if (( $? == 0 )); then
    echo "found"
else
    echo "not found"
fi
```

```
if grep -q "^admin:" /etc/passwd
then
    echo "found"
else
    echo "not found"
fi
```

&& 和 ||

```
if [[ -z $default ]]; then  
    default="ok"  
fi
```

```
[[ -z $default ]] && default="ok"  
[[ -n $default ]] || default="ok"
```

&& 和 ||

```
if grep admin /etc/passwd; then
    echo "found"
else
    echo "not found"
fi
```

```
grep admin /etc/passwd &&
    echo "found" ||
    echo "not found"
```

&& 和 ||

```
if [[ $v != "yes" ]] && [[ $v != "y" ]]  
then
```

```
.....
```

```
fi
```

```
if [[ $v == "ok" ]] || [[ $v == "yes" ]]  
then
```

```
.....
```

```
fi
```

case

```
## 语法
```

```
case $var in
    condition1)
        ...
        ;;
    condition2)
        ...
        ;;
*)
    ...
    ;;
esac
```

```
read answer
case $answer in
    yes)
        echo yes
        ;;
    no)
        echo no
        ;;
*)
    echo unknown
    ;;
esac
```

case 中的字符匹配

- * 匹配零个或多个任意字符
- ? 匹配一个任意字符
- [] 字符组，可以匹配组中的一个字符
- | A|B, A 或者 B

case 中的字符匹配

```
case $answer in
  [Yy][Ee][Ss])
    echo "yes"
    ;;
  [Nn][Oo])
    echo "no"
    ;;
  *)
    echo "unknown"
    ;;
esac
```

```
case $answer in
  yes|y)
    echo "yes"
    ;;
  no|n)
    echo "no"
    ;;
  *)
    echo "unknown"
    ;;
esac
```

while 循环

形式 1

```
while expression; do
    statements
done
```

形式 2

```
until expression; do
    statements
done
```

#example

```
i=1
```

```
sum=0
```

```
while (( i <= 100 )); do
```

```
    (( sum += i ))
```

```
    (( i++ ))
```

```
done
```

```
echo sum: $sum
```


统计行数

```
#!/bin/bash
file=$0
cnt=0
while read; do
    (( cnt ++ ))
    echo -e "$cnt:\t$REPLY"
done < $file          # 关键

echo "total:[$cnt]"    # finally
```

for 循环

```
for var in var1 var2 ...  
do  
    statments  
done
```

break 中断循环
continue 继续循环

```
# example  
for file in *.html  
do  
    if [[ -h $file ]]; then  
        echo "$file is link"  
    fi  
done
```

重定向

- 一个进程运行时，内核为其准备三个默认文件描述符
- STDIN (0), STDOUT(1), STDERR (2)

```
echo xxx > file
echo xxx 1> file      # 太土
echo xxx >> file
read line < file
some_com2 > log.err
some_com > /dev/null 2>&1
some_com &> /dev/null # 推荐
some_com >> log 2>&1
```

块重定向 - 1

```
if [[ $error ]]; then
    echo "error: $error" >> "$log"
else
    echo "ok" >> "$log"
fi
```

```
if [[ $error ]]; then
    echo "error: $error"
else
    echo "ok"
fi >> "$log"
```

块重定向 - 2

```
echo message 1 >> "$log"  
echo message 2 >> "$log"
```

```
{  
    echo message 1  
    echo message 2  
} >> "$log"
```

```
(  
    echo message 1  
    echo message 2  
) >> "$log"
```

块重定向

- 代码块：
 - if , case 块
 - for, while 循环体
 - { }, () 命令分组
- 块中共享相同的文件描述符（即统一重定向）
- 除非在块中自行打开、或关闭

函数

- 调用： 需要先定义后调用。名字 + 参数
- 参数： \$1, \$2 ... \$#, \$@
- 返回值： 标准输出
- 退出状态： `return $num` ; 或最后一个命令的退出状态
- 注意
 - 函数必须先定义，后使用

函数

```
# define  
func_name()  
{  
    ...  
}
```

local 可指定局部变量

```
# example  
add()  
{  
    local sum  
    (( sum = $1 + $2 ))  
    echo $sum  
}  
  
cnt=$( add 10 20 )
```


判断 OS

```
is_suseos()  
{  
    if [[ -f /etc/SuSE-release ]]  
    then  
        return 0  
    else  
        return 1  
    fi  
}
```

```
is_suseos()  
{  
    [[ -f /etc/SuSE-release ]]  
}
```

调用：

```
if is_suseos; then  
    echo "SUSE OS"  
else  
    echo "Slackware ?"  
fi
```

调用一个函数，与调用一个命令是完全一致的


函数的命名

- 动作类：
 - get_xxx , 如 get_lanlip
 - set_xxx , 如 set_hostname
- 判断真假类：
 - is_xxx , 如 is_suseos , is_app_ok
 - has_xxx , 如 has_ip

关于函数的建议

- 函数名要具可读性，从名字即能看出功能
- 函数尽量精练，只完成单一功能
- 尽量使用局部变量 (`local val`)
- 尽量在 `$?` 中返回失败 / 成功
- 调用函数时要尽量检查其返回值
- 一般不需要编写 `main()`

内容导航

- Bash 基础 ✓
- Bash 语法 ✓
- 监控实例 

监控实例

- 进程监控的两种常用方法：
 - crontab 周期性调度，执行检查
 - 独立运行，循环检查
- 例： `cron_mon.sh`， 一个脚本的成长过程

关于脚本的建议

- 代码应分三部分：变量，函数，主程序
- 给变量和函数起一些有意义的名字，适当注释
- 变量多时，可独立成配置文件
- 特定功能，用函数实现
- 函数多时，可独立成函数库
- 酌情记录日志
- 严重错误时要告警

Bash 脚本的调试

- 运行前, `bash -n` 检查语法
- 用 `bash -x` 打印执行过程

内容导航

- Bash 基础 ✓
- Bash 语法 ✓
- 监控实例 ✓

推荐书目

- 经典资料
 - 《abs-guide》推荐
 - 《unix shell 范例精解》
 - 《Sed 与 awk》
 - 《精通正则表达式》
 - **man-pages**
- 关于编程风格
 - 《Perl 最佳实践》
 - 《Perl Style Guide》文章

Q & A

Thanks !

欢迎访问 Bash K 吧

<http://km.oa.com/group/bash>