Module01-02
Linux 基础：深入了解 bash

Linux 基础 - 深入了解 bash

- 常用 Linux 命令
- ➔ 深入了解 bash
- 正则表达式基础
- find 、 grep 、 sed 、 awk

深入了解 bash - bash 特性

- Bash 的一些特性
  - I/O 重定向
  - 管道（ pipe ）
  - 可使用通配符（如 * 、 ? 等）来匹配文件名
  - 支持命令历史（ history 和 fc ）
  - 支持命令别名
  - 支持文件名（文件、路径、命令名）补全（ tab 键）
  - 任务控制
  - 支持命令行编辑
  - 可使用 shell 变量或选项定制 shell 环境
  - 众多的 shell 内置命令，方便 shell 编程
  - 支持 shell 函数，使 shell 编程模块化
  - 支持算术表达式

深入了解 bash - Shell 语法

- 常用 Linux 命令
- 深入了解 bash
  - Shell 语法
    - 变量
    - 任务控制（ job control ）
- 正则表达式基础
- find 、 grep 、 sed 、 awk

深入了解 bash - Shell 语法

- Shell 语法
  - 几个特别的文件
  - 文件名通配符（元字符）
  - 特殊字符
  - 命令形式
  - I/O 重定向形式

Shell 语法 - 几个特别的文件

- 几个特别的文件

  shell 进程启动后，首先会去读取并执行几个文件中的命令：

  - 登录后首先自动执行： /etc/profile
  - 其次依次查找并自动执行： ~/.bash_profile, ~/.bash_login，或 ~/.profile
  - 然后就是： ~/.bashrc

Shell 语法 - 文件名通配

- 文件名通配符（一些元字符）
  - 常用的通配符

字符	意思
*	匹配任意字符数（0到多个）的字符串
?	匹配任意单个字符
[abc...]	匹配括号内列举的任意一个字符，如：[abc4]，匹配字符a、b、c、4其中一个 -代表范围，如[a-d]，匹配a、b、c、d其中的一个
[!abc...]	与上行相反，凡是出现在[ ]中的字符都不匹配，即匹配除[]中出现的字符以外的所有单个字符
~	当前用户的主目录
~user	用户user的主目录
~+	当前工作目录($PWD)
~-	之前工作目录($OLDPWD)

Shell 语法 - 文件名通配

- 文件名通配符（一些元字符）（续）
  - 示例

```
$ ls c*.sql
cb.sql  cc.sql
$ ls c[bc].sql
cb.sql  cc.sql
$ ls c[a-c].sql
cb.sql  cc.sql
$ ls c[!c-z].sql
cb.sql
$ ls nf*
nf1  nf12  nf2
$ ls nf?
nf1  nf2
$ ls nf[1-9]
nf1  nf2
```

Shell 语法 - 特殊字符

- 特殊字符

字符	意思
;	命令分隔
&	后台执行
()	命令组
|	管道（ pipe ）
< > &	I/O 重定向符号
* ? [ ] ~ + - @ !	文件名元字符
" ' \	转义字符
`	命令替代
$	变量替代
空格、 tab 、新行	单词分隔

Shell 语法 - 特殊字符

- 一些说明：
  - “ ”（双引号）和' '（单引号）：
    - 凡是包含在 ” ” 或 ' ' 之间的内容都被解释为字符；
    - 双引号和单引号的区别：出现在双引号内的变量会被展开（解释），而出现在单引号之间的变量不会被解释，如：

```
$ echo "Tiger's home is $HOME"
Tiger's home is /home/kwarph
$ echo 'My home is $HOME'
My home is $HOME
```

    - 要在双引号之间包含：”、 $ 作为普通，需要转义如：\”， \$

Shell 语法 - 特殊字符

- 示例：

```
$ name=tiger
$ echo "the value of \$name is $name"
the value of $name is tiger
$ echo "current path: `pwd`"
current path: /home/kwarph/linux_cmd/files
```

Shell 语法 - 命令形式

- 命令形式

命令	效果
cmd &	后台执行命令 cmd
cmd1 ; cmd2	按次序执行 cmd1,cmd2，不论前面的命令是否执行成功
{ cmd1 ; cmd2 ; }	作为命令组在当前 shell 执行
(cmd1 ; cmd2)	作为命令组在当前 shell 的子 shell 下执行
cmd1 | cmd2	管道，前一个命令的输出作为后一个命令的输入
cmd1 `cmd2`	命令 cmd1 用命令 cmd2 的输出作为参数
cmd1 $(cmd2)	同上，POSIX 标准
cmd $((expression))	将表达式 expression 的结果作为 cmd 的参数，POSIX 标准
cmd1 && cmd2	“与”，按次序执行，且只有前面的命令执行成功，才执行后续的命令
cmd1 || cmd2	“或”，如果 cmd1 执行成功，不用执行 cmd2，只有 cmd1 执行不成功，才执行 cmd2
! cmd	“非”，反转 cmd 执行结果（只适用于 shell 脚本） 注：!cmd 在终端执行表示调用历史命令 cmd

Shell 语法 - 命令形式

- 示例：

```
$ sort file > sort.dat &          # 后台排序文件，将结果输出到 sort.dat
$ pwd; ls                         # 先执行 pwd，后执行 ls
$ (date; who; pwd) > logfile      # 将前面命令组中每个命令的输出输出到 logfile
$ sort file | grep kwarph         # 管道
$ vi `grep -l ifdef *.c`          # 将 grep 找到的文件作为 vi 编辑的文件名
$ egrep '(yes|no)' `cat list`          # 同上
$ egrep '(yes|no)' $(cat list)         # 同上，POSIX 标准
$ egrep '(yes|no)' $(< list)           # 同上，更快，但不是 POSIX 标准
$ grep XX file && lp file         # 如果 grep 在 file 中找到 XX，则打印 file
$ grep XX file || echo "XX not found"   # 如果在 file 未找到 XX，则输出”
XX not found”
```

Shell 语法 - I/O 重定向形式

- I/O 重定向形式
  - 说明：
    - UNIX/Linux 下一切皆文件
    - 每个开启的文件都有一个文件描述符（ file description ，一个正整数）与之关联，获取文件描述符就可以对文件进行读写操作
    - 三个预定义的文件：
      - 标准输入（ stdin，通常为键盘），文件描述符为 0
      - 标准输出（ stdout，通常为屏幕），文件描述符为 1
      - 标准错误（ stderr，通常为屏幕），文件描述符为 2

Shell 语法 - I/O 重定向形式

- I/O 重定向形式（续）
  - 简单 I/O 重定向：
    - cmd > file （ cmd :> file ）

      将 cmd 命令的输出写入文件 file，如 file 存在，清空之，不存在则创建之

      `>file` 或 `:>file` 清空或创建文件 file
    - cmd >> file （ cmd :>> file ）

      将 cmd 命令的输出追加到文件 file 末尾，如 file 不存在，则创建之

      `>>file` 或 `:>>file` `file` 不存在则创建之，存在则无改变
    - cmd < file

      cmd 命令从文件 file 获取输入
    - cmd << text

      命令 cmd 从标准输入（键盘）获取输入，直到遇到 text 结束输入，很多命令都使用这种方式，如： cat << EOF， text 被称为“Here String”
    - cmd <> file

      以可读、可写的方式打开文件 file

Shell 语法 - I/O 重定向形式

- I/O 重定向形式（续）
  - 使用文件描述符的 I/O 重定向：

    表中的 m 和 n 代表文件描述符

语法	效果
cmd >&n	将命令 cmd 的输出定向到文件描述符 n，如 cmd >&2
cmd m>&n	同上，同时也将原本要定向到 m 的内容也定向到 n，如：cmd 2>&1，表示不论标准输出、标准错误，都定向到标准输出
cmd >&-	关闭标准输出
cmd <&n	从文件描述符 n 获取输入
cmd m<&n	同上，同时原本需从 m 输入的，也从 n 输入
cmd <&-	关闭标准输入
cmd <&n-	类似 cmd <&n，但不复制文件描述符 n
cmd >&n-	类似 cmd >&n，但不复制文件描述符 n

Shell 语法 - I/O 重定向形式

- I/O 重定向形式（续）
  - 多重 I/O 重定向：

语法	效果
`cmd 2>file`	将标准错误定向到 `file`，但标准输出不受影响
`cmd > file 2>&1`	不论标准输出、标准错误，都定向到文件 `file`
`cmd &> file`	同上，只适用于 `bash`，推荐的形式
`cmd >& file`	同上，只适用于 bash
`cmd > f1 2>f2`	标准输出定向到文件 `f1`，标准错误定向到 `f2`
`cmd | tee files`	将 cmd 的输出同时定向到标准输出（屏幕）和文件 `files`
`cmd 2>&1 | tee files`	将标准输出、标准错误同时定向到标准输出（屏幕）和文件 files

Shell 语法 - I/O 重定向形式

- I/O 重定向形式（续）
  - 示例：

```
$ cat cb.sql > users.sql
$ cat ca.sql cc.sql >> users.sql
$
$ tr 'cm' 'Gb' < ca.sql
$
$ cat << EOF
> the first line.
> the second line.
> last line, see 'here string - EOF'
> EOF
the first line.
the second line.
last line, see 'here string – EOF'
$ /home/kwarph/Im/bin/ImServer > /dev/null 2>&1 &
```

深入了解 bash - 变量

- 常用 Linux 命令
- 深入了解 bash
  - Shell 语法
  - 变量
  - 任务控制（ job control ）
- 正则表达式基础
- find 、 grep 、 sed 、 awk

深入了解 bash - 变量

- 变量
  - 变量替代
  - Shell 内置变量
  - 其它一些变量
  - 数组
  - 特殊的提示字符串

变量 - 变量替代

- 变量替代

表达式	描述
var=value	设置变量 var 的值为 value
${var}	取变量 var 的值，大括号在不混淆的情况下可省
${var:-value}	取变量 var 的值，如 var 没设置，则取值 value
${var:=value}	取变量 var 的值，如 var 没设置，则取值 value，且设置 var 值为 value
${var:?value}	取 var 的值，如果 var 没设置，打印 value ，退出
${var:+value}	如果 var 已设置，取 value ，否则不取
${#var}	变量 var 的长度
${#*}	取所有的位置参数
${#@}	取所有的位置参数
${var#pattern}	移除 var 值中匹配样式的左边的片段，移除最少的
${var##pattern}	移除 var 值中匹配样式的左边的片段，移除最多的
${var%pattern}	移除 var 值中匹配样式的右边的片段，移除最少的
${var%%pattern}	移除 var 值中匹配样式的右边的片段，移除最多的

- 变量替代（续）
  - 示例：

```
$ u=up d=down blank=
$ echo ${u}root    # 注意：这里必须使用 { } 来取变量 u 的值
uproot
$ echo ${u-$d}     # 取变量 u 的值，如果 u 没设置，则取 d 的值
up
$ echo ${tmp-`date`}   # 取变量 tmp 的值，如果 tmp 没设置，则执行 date
2009 年 11 月 24 日 星期二 16:52:40 CST
$ echo ${blank="no data"}   #blank 之前没设置，所以打印空行
$ echo ${blank:="no data"}
no data
$ echo $blank
no data
$ pwd
/home/kwarph/linux_cmd/files
$ tail=${PWD##*/}
$ echo $tail
files
```

变量 - Shell 内置变量

- Shell 内置变量
  - 常用的内置变量

变量	描述
$#	命令行参数的个数
$-	当前生效的选项
$?	最近一次执行命令的退出状态（退出值）
$$	当前进程的进程号
$!	最近一个后台任务的进程号
$0	命令或脚本的名称
$n	第 n 个命令行参数，如果 n>9 ，则需使用大括号：${12}
$*, $@	所有命令行参数（$1 $2 $3...）
"$*"	用一行表示所有命令行参数（“$1 $2 $3...”）
"$@"	所有命令行参数，用引号分别隔开（“$1” “$2” “$3” ...）

变量 - Shell 内置变量

- 示例：

```
$ cat argvs.sh
#!/bin/bash
echo $0 with $# args: "$*"   # 打印命令名、参数个数、所有参数
index=1
for arg in $*      # 也可以用: for arg in “$@”
do
    echo "Argument ${index} is: ${arg}"
    let "index+=1"
done
exit 0
$ ./argvs.sh tiger lion
./argvs.sh with 2 args: tiger lion
Argument 1 is: tiger
Argument 2 is: lion
$ echo $?        # 查看最后一次执行的命令 argvs.sh 的退出状态码
0
$ echo $$        # 查看当前进程号
5639
```

变量 - 其它一些变量

- 其它一些变量

变量	描述
SHELL	shell 名称
PWD	当前工作目录（当前目录）
HOME	当前用户的主目录
PATH	可执行文件搜索路径，多个路径之间用冒号（:）分隔
LD_LIBRARY_PATH	库文件搜索路径，多个路径之间用冒号（:）分隔
IFS	输入字段的分隔符，如多个选项、参数之间的空白字符
PS1	shell 主提示符，如 bash 的提示符， root 为 # ，普通用户为 $
PS2	shell 第二提示符，默认为 >（如 cat 命令从键盘输入时的提示符）
PS3	在循环中选择时的提示符，默认为 #?
PS4	用于 shell 脚本的 debug 时的提示符，跟踪脚本调用的每个命令，默认为 +
LINENO	记录 shell 脚本的执行的当前行行号

- 其它一些变量（续）
  - 用于 PS1 、 PS3 、 PS4 的格式字符

字符	描述	字符	描述
\a	ASCII BEL 字符 (\007)	\A	当前时间， 24 小时格式 HH:MM
\d	当前日期的“周 月 日”格式	\D {fmt}	格式化显示当前日期
\e	脱字符 (\033)	\h 、 \H	主机名、完整主机名
\[、 \]	格式控制起止字符	\n	新行
\r	回车符	\s	shell 名称
\t	当前时间， 24 小时格式	\T	当前时间， 12 小时格式
\u	用户名	\v	bash 的版本
\w	当前路径	\W	当前目录
\$	EUID 为 0 则 # ，否则 $	\@	当前时间， 12-hour a.m./p.m.
\nnn	字符的八进制表示	\\	\

变量 - 其它一些变量

- 其它一些变量（续）
  - 示例：

```
kwarph@xuanyuan-soft:~/linux_cmd$ echo $PS1
\[\e]0;\u@\h: \w\a\]\u@\h:\w$
kwarph@xuanyuan-soft:~/linux_cmd$ OLD_PS1=$PS1
kwarph@xuanyuan-soft:~/linux_cmd$ PS1="\s\v-\w$ "
bash3.2-~/linux_cmd$                    # 注意提示符的变化
bash3.2-~/linux_cmd$ PS1=$OLD_PS1
kwarph@xuanyuan-soft:~/linux_cmd$       # 提示符又换回来了
```

变量 - 其它一些变量

- 其它一些变量（续）
  - 关于 PS3 的示例

```
$ cat ps3_test.sh
#!/bin/bash
#  如果不设 PS3 的值，提示符将是 #?，而不是下
#  面的 Select an operation:
PS3='Select an operation: '
select i in Time Info Users Quit
do
    case $i in
        Time) date;;
        Info) uname -a;;
        Users) w;;
        Quit) break;;
    esac
done
```

```
$ ./ps3_test.sh
1) Time
2) Info
3) Users
4) Quit
Select an operation: 1      # 注意提示符是一句话
2009 年 11 月 25 日 星期三 11:28:31 CST
Select an operation: 4
$
```

变量 - 其它一些变量

- 其它一些变量（续）
  - 设置 PATH 和 LD_LIBRARY_PATH
    - 仅在本次会话中生效：

```
$ export PATH="/home/kwarph/bin:$PATH"
$ export LD_LIBRARY_PATH="/home/kwarph/lib:$LD_LIBRARY_PATH"
```

    - 保存上述环境变量到 ~/.bash_profile（只针对当前用户）
    - 保存上述环境变量到 /etc/profile（需 root 权限，针对所有用户）

（注：上述 2 个变量，在开发环境配置课程中会有详细的讲解）

深入了解 bash - 任务控制

- 常用 Linux 命令
- 深入了解 bash
  - Shell 语法
  - 变量
  - 任务控制（ job control ）
- 正则表达式基础
- find 、 grep 、 sed 、 awk

深入了解 bash - 任务控制

- 任务控制
  - 说明：任务控制是指将一个任务（进程）放到后台、或将后台任务提到前台，以及挂起（暂停）等
  - 通常的操作：

操作	描述	操作	描述
cmd &	后台执行命令 cmd	%n	号码为 n 的任务
%s	名为 s 的任务	%?s	名称包含字符串 s 的任务
%%、 %+	当前任务	%-	前一个任务
bg	将任务放到后台	fg	将任务提到前台
jobs	列出激活的任务	kill	终止任务（进程）
suspend	挂起当前 shell	wait	等待所有后台任务结束
Ctrl-Z	挂起当前任务。挂起的任务可以通过 fg、 bg 继续		

深入了解 Bash - 总结

通过本单元的学习：

- 我们熟悉了 bash 命令行的一些格式：如命令的形式、文件名通配、 I/O 重定向等
- 了解 Shell 变量规则以及一些有用的变量，为日后的 shell 编程打下基础
- 了解任务控制的方法