shell基础知识

教师: 李军辉

管理整个计算机硬件的其实是操作系统的核心(kernel)，这个核心是需要被保护的！

那用户如何通过操作系统的核心来使用计算机硬件?

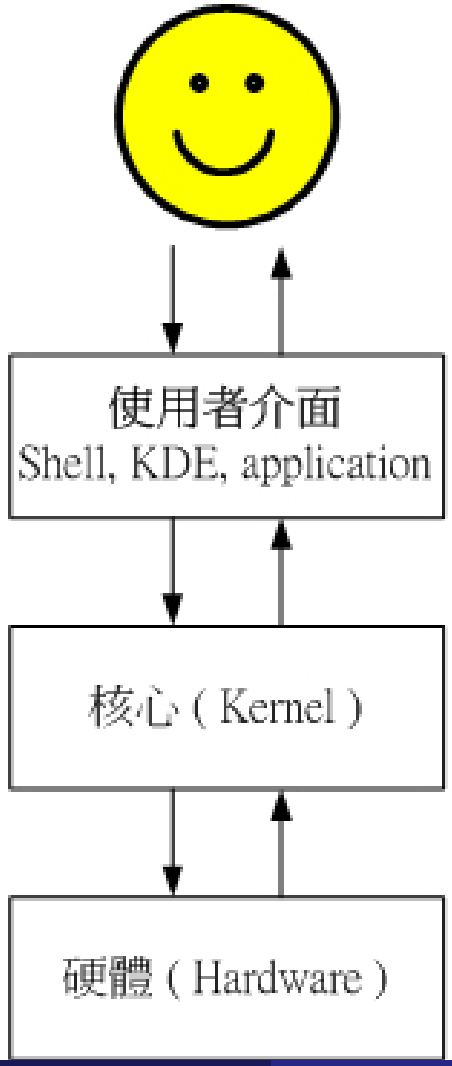
find -name 1.txt /

find -name 1.txt /

1. 用户输入命令
2. 硬件执行命令

find -name 1.txt /

1. 用户输入命令
2. 硬件执行命令



您就是這個可愛的笑臉，
使用文字或圖形介面，
在螢幕之前操作你的作業系統。

接受來自使用者的指令，
以與核心進行溝通。

真正在控制硬體工作的哆哆
含有 CPU 排程、記憶體管理、
磁碟輸出輸入等工作。

整個系統中的實體工作者，
包含了硬碟、顯示卡、網路卡、
CPU、記憶體等等。

壳程序 (Shell)

壳程序的功能只是提供用户操作系统的一个接口。

命令（如ls, cp, mkdir等）都是独立的应用程序，但是我们可以透过壳程序 (就是命令列模式) 来操作这些应用程序，让这些应用程序呼叫核心来运行所需的工作。

也就是说，只要能够操作应用程序的接口都能够称为壳程序。狭义的壳程序指的是命令列方面的软件，包括本章要介绍的 bash 等。广义的壳程序则包括图形接口的软件！因为图形接口其实也能够操作各种应用程序来呼叫核心工作啊！不过在本章中，我们主要还是在使用 bash 啦！

为何要学文字接口的Shell

为何要学文字接口的Shell

1. 文字接口的 shell：大家都一样！
2. 远程管理：文字接口就是比较快！

Linux 有哪些Shell? 我们使用的是哪一个 Shell ?

cat /etc/shells

cat /etc/passwd

Bash

- Bash， UNIX Shell的一种， 在1987年由布莱恩·福克斯(Brian Fox)为了GNU计划而编写。
- 1989年发布第一个正式版本，原先是计划用在GNU操作系统上，但能运行于大多数类Unix系统的操作系统之上，包括Linux与Mac OS X v10.4都将它作为默认shell。

Bash

- sh: 较bash之前的一个shell, 创始人是Steven Bourne.
- 为Bourne shell的后继兼容版本与开放源代码版本，它的名称来自Bourne shell（sh）的一个双关语（Bourne again / born again）：Bourne-Again SHell。
- Bash是一个命令处理器，通常运行于文本窗口中，并能执行用户直接输入的命令。
- Bash还能从文件中读取命令，这样的文件称为脚本。

Bash

- 提示符
  - $
  - #
- lijunhui@lijunhui-OptiPlex-9020:~/workspace/pdtb-pyn/src$
- 命令一般由三个部分组成
  - 命令
  - 选项
  - 参数
- mkdir -p ~/workspace/test/src

/bin/bash 是 Linux 默认的 shell

bash 是 GNU 计划中重要的工具软件之一

bash 主要的优点:

1. 命令编修能力 (history)
2. 命令与文件补全功能： ([tab] 按键的好处)
3. 命令别名配置功能
4. 工作控制、前景背景控制
5. 程序化脚本
6. 通配符

命令编修能力 (history)

- 按『上下键』就可以找到前/后一个输入的命令！
- 文件~/.bash_history

Bash: 历史记录

通过历史记录简化操作

- !! 重复前一个命令
- !字符 重复前一个以‘字符’开头的命令
- !num 按照历史记录的序号执行命令
- !?abc 重复之前包含’abc’的命令
- !-n 重复前第n个命令

Bash: 历史记录

重复Bash 历史记录操作

- 通过 Ctrl+R 在历史记录中搜索命令
- 重新调用前一个命令中的参数 按esc之后再按.

命令与文件补全功能

Tab 接在一串命令的第一个字的后面，则为命令补全；

Tab 接在一串命令的第二个字以后时，则为『文件补齐』

命令别名配置功能

给命令配置快捷方式.

- alias lm='ls -al'

命令别名配置功能

如何将linux下的rm命令改造成移动文件至回收站?

命令别名配置功能

如何将linux下的rm命令改造成移动文件至回收站?

步骤1

mkdir -p ~/.trash
vim ~/.bashrc

命令别名配置功能

如何将linux下的rm命令改造成移动文件至回收站?

步骤1

```
mkdir -p ~/.trash
vim ~/.bashrc
```

步骤2

```
alias rm='trash'
trash()
{
    mv $@ ~/.trash/
}
```

工作控制、前景背景控制

工作控制、前景背景控制： (job control, foreground, background)

工作控制、前景背景控制

- 在后台运行进程
  - 在命令后添加一个&
- 暂停某个程序
  - Ctrl+Z
- 管理后台作业
  - jobs
  - bg
  - fg

程序化脚本： (Shell scripts)

Shell编程

Bash: 通配符 (Wildcard)

Bash Shell支持以下通配符

- * 匹配0个或多个
- ? 匹配任意一个字符
- [0-9] 匹配一个数字范围
- [abc] 匹配列表里任何字符
- [âbc] 匹配列表里以外字符

Shell 的变量功能

- 变量基本操作
- 双引号/单引号/反撇号
- 常见的环境变量
- 其他特殊变量

变量的基本操作

定义及赋值

- 格式: 变量名=变量值

注意: 等号前后不能存在空格

变量的基本操作

定义及赋值

- 格式: 变量名=变量值

注意: 等号前后不能存在空格 引用变量

- 格式: $变量名、${变量名}

变量的基本操作 (示例1)

```
MyName=Zhangsan

echo $MyName
```

三种定界符号: 双引号、单引号、反撇号

双引号:

- 允许引用、转义

单引号:

- 禁止引用、转义

反撇号, 或者$()

- 以命令输出进行替换

三种定界符号 (示例1)

MyName=Zhangsan

echo “My name is $MyName”

echo ‘My name is $MyName’

三种定界符号 (示例2)

uname -r

Ver=‘uname -r‘

echo $Ver

常见的环境变量

用来记录、设置运行参数

- 系统赋值: USERNAME, LOGNAME, HOME, SHELL, ...
- 用户操作: PATH, LANG, CLASSPATH, ...

环境变量的功能

1. env
2. set
3. export

env

观察环境变量与常见环境变量说明.

- HOME: 代表用户的家目录。
- SHELL: 目前这个环境使用的 SHELL 是哪个程序. Linux 默认使用 /bin/bash
- HISTSIZE: 我们曾经下达过的命令可以被系统记录下来，而记录的『笔数』则是由这个值来配置的。

env

- MAIL: 使用 mail 这个命令在收信时，系统会去读取的邮件信箱文件。
- PATH: 就是运行文件搜寻的路径～目录与目录中间以冒号(:)分隔，由于文件的搜寻是依序由 PATH 的变量内的目录来查询，所以，目录的顺序也是重要的。

set

观察所有变量 (含环境变量与自定义变量).

基本上，在 Linux 默认的情况中，使用大写的字母来配置的变量一般为系统内定需要的变量.

env 与 set 的区别?

export

自定义变量转成环境变量

其他特殊变量

由系统或脚本控制, 不能直接赋值.

- $?: 前一条命令的状态值: 0为正常, 非0为异常.
- $0: 脚本自身的程序名.
- $1 - $9: 第1 - 第9个位置参数
- $*: 命令行的所有位置参数内容
- $#: 命令行的位置参数个数

其他特殊变量 (示例1)

mkdir /mydir

echo $?

其他特殊变量 (示例1)

mkdir /mydir

echo $?

其他特殊变量 (示例2)

test.sh

1. 若该变量需要在其他子程序运行，则需要以 export 来使变量变成环境变量.
2. 通常大写字符为系统默认变量，自行配置变量可以使用小写字符，方便判断
3. 取消变量的方法为使用 unset

范例一：配置一变量 name，且内容为 VBird

```
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found
[root@www ~]# name  =  Bird
[root@www ~]# name=VBird
```

范例二：承上题，若变量内容为 VBird's name 呢，就是变量内容含有特殊符号时：

[root@www ~]# name=VBird's name

单引号与双引号必须要成对，在上面的配置中仅有一个单引号

你还可以继续输入变量内容。这与我们所需要的功能不同，失败啦！

记得，失败后要复原请按下 [ctrl]-c 结束！

[root@www ~]# name="VBird's name"

命令是由左边向右找，先遇到的引号先有用，因此如上所示，单引号会失效！

[root@www ~]# name='VBird's name'

范例三：在 PATH 这个变量当中『累加』:/home/dmtsai/bin

[root@www ~]# PATH=$PATH:/home/dmtsai/bin

[root@www ~]# PATH="$PATH":/home/dmtsai/bin

[root@www ~]# PATH=$PATH:/home/dmtsai/bin

范例四：承范例三，将name的内容多出“yes”呢？

[root@www ~]# name=$nameyes

知道了吧？如果没有双引号，那么变量成了啥？name 的内容是 $nameyes 这个变量！

呵呵！我们可没有配置过 nameyes 这个变量！所以，应该是底下这样才对！

[root@www ~]# name=“$name”yes

[root@www ~]# name=$nameyes

范例五：如何让刚刚配置的 name=VBird 可以用在下个 shell 的程序？

```
[root@www ~]# name=VBird
[root@www ~]# bash
[root@www ~]# echo $name
[root@www ~]# exit
[root@www ~]# export name
[root@www ~]# bash
[root@www ~]# echo $name
[root@www ~]# exit
```

范例六：如何进入到您目前核心的模块目录?

[root@www ~]# cd /lib/modules/‘uname -r‘/kernel

[root@www ~]# cd /lib/modules/$(uname -r)/kernel

范例七：取消刚刚配置的 name 这个变量内容

[root@www ~]# unset name

问题1

在变量的配置当中，单引号与双引号的用途有何不同？

问题1

在变量的配置当中，单引号与双引号的用途有何不同？

问题1

在变量的配置当中，单引号与双引号的用途有何不同？

回答

单引号与双引号的最大不同在于双引号仍然可以保有变量的内容，但单引号内仅能是一般字符，而不会有特殊符号。

```
[root@www ~]# name=VBird
[root@www ~]# echo $name
VBird
[root@www ~]# myname="$name its me"
[root@www ~]# echo $myname
VBird its me
```

问题2

在命令下达的过程中，反单引号(‘)这个符号代表的意义为何?

问题2

在命令下达的过程中，反单引号(‘)这个符号代表的意义为何?

回答

在一串命令中，在‘之内的命令将会被先运行，而其运行出来的结果将做为外部的输入信息!

ls -l ‘locate gcc‘

问题3

若你有一个常去的工作目录名称为：『/cluster/server/work/taiwan_2005/003/』，如何进行该目录的简化?

问题3

若你有一个常去的工作目录名称为：『/cluster/server/work/taiwan_2005/003/』，如何进行该目录的简化?

回答

在一般的情况下，如果你想要进入上述的目录得要『cd /cluster/server/work/taiwan_2005/003/』

```
work="/cluster/server/work/taiwan_2005/003/"
cd $work
```

变量内容的删除、替代与替换

```
范例一：先让小写的 path 自定义变量配置的与 PATH 内容相同
[root@www ~]# path=${PATH}
[root@www ~]# echo $path
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin   <==这两行其实是同一行啦！

范例二：假设我不喜欢 kerberos，所以要将前两个目录删除掉，如何显示？
[root@www ~]# echo ${path#/*kerberos/bin:}
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

```
${variable#/*kerberos/bin:}
  上面的特殊字体部分是关键词！用在这种删除模式所必须存在的

${variable#/*kerberos/bin:}
  这就是原本的变量名称，以上面范例二来说，这里就填写 path 这个『变量名称』啦！

${variable#/*kerberos/bin:}
  这是重点！代表『从变量内容的最前面开始向右删除』，且仅删除最短的那个

${variable#/*kerberos/bin:}
  代表要被删除的部分，由于 # 代表由前面开始删除，所以这里便由开始的 / 写起。
  需要注意的是，我们还可以透过通配符 * 来取代 0 到无穷多个任意字符

  以上面范例二的结果来看， path 这个变量被删除的内容如下所示：
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin   <==这两行其实是同一行啦！
```

范例三：我想要删除前面所有的目录，仅保留最后一个目录

```
[root@www ~]# echo ${path#/*:}
/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
/root/bin       <==这两行其实是同一行啦！
# 由于一个 # 仅删除掉最短的那个，因此他删除的情况可以用底下的删除线来看：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==这两行其实是同一行啦！

[root@www ~]# echo ${path##/*:}
/root/bin
# 嘿！多加了一个 # 变成 ## 之后，他变成『删除掉最长的那个数据』！亦即是：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:/root/bin  <==这两行其实是同一行啦！
```

范例四：我想要删除最后面那个目录，亦即从 : 到 bin 为止的字符串

```
[root@www ~]# echo ${path%:*bin}
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin   <==注意啊！最后面一个目录不见去！
# 这个 % 符号代表由最后面开始向前删除！所以上面得到的结果其实是来自如下：
# /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
# /usr/sbin:/usr/bin:~~/root/bin~~  <==这两行其实是同一行啦！
```

范例五：那如果我只想要保留第一个目录呢？

```
[root@www ~]# echo ${path%%:*bin}
/usr/kerberos/sbin
# 同样的， %% 代表的则是最长的符合字符串，所以结果其实是来自如下：
# /usr/kerberos/sbin~~:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:~~
# ~~/usr/sbin:/usr/bin:/root/bin~~  <==这两行其实是同一行啦！
```

范例六：将 path 的变量内容内的 sbin 取代成大写 SBIN：

```
[root@www ~]# echo ${path/sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin
# 这个部分就容易理解的多了！关键词在于那两个斜线，两斜线中间的是旧字符串
# 后面的是新字符串，所以结果就会出现如上述的特殊字体部分啰！

[root@www ~]# echo ${path//sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/SBIN:/usr/local/bin:/SBIN:/bin:
/usr/SBIN:/usr/bin:/root/bin
# 如果是两条斜线，那么就变成所有符合的内容都会被取代喔！
```

1. ${变量#关键词}
   - 若变量内容从头开始的数据符合『关键词』，则将符合的最短数据删除

2. ${变量##关键词}
   - 若变量内容从头开始的数据符合『关键词』，则将符合的最长数据删除

1. ${变量%关键词}
   - 若变量内容从尾向前的数据符合『关键词』，则将符合的最短数据删除
2. ${变量%%关键词}
   - 若变量内容从尾向前的数据符合『关键词』，则将符合的最长数据删除

1. ${变量/旧字符串/新字符串}
   - 若变量内容符合『旧字符串』则『第一个旧字符串会被新字符串取代』
2. ${变量//旧字符串/新字符串}
   - 若变量内容符合『旧字符串』则『全部的旧字符串会被新字符串取代』

Bash Shell的操作环境

命令的运行顺序:

1. 以相对/绝对路径执行命令，例如“/bin/ls”或‘./ls’
2. 由alias找到该命令来执行
3. 由bash内置的(builtin)命令来执行
4. 通过$PATH这个变量的顺序找到第一个命令来执行

Linux Shell中的特殊符号

- * (代表零个或多个任意字符)
- ? (只代表一个任意的字符)
- # (注释符号)
- \(脱义字符)

Linux Shell中的特殊符号

- |(管道符)
- $
- ;
- ~

Linux Shell中的特殊符号

- &
- >、>>、2>和2>> (重定向符号)
- []
- &&
- ||