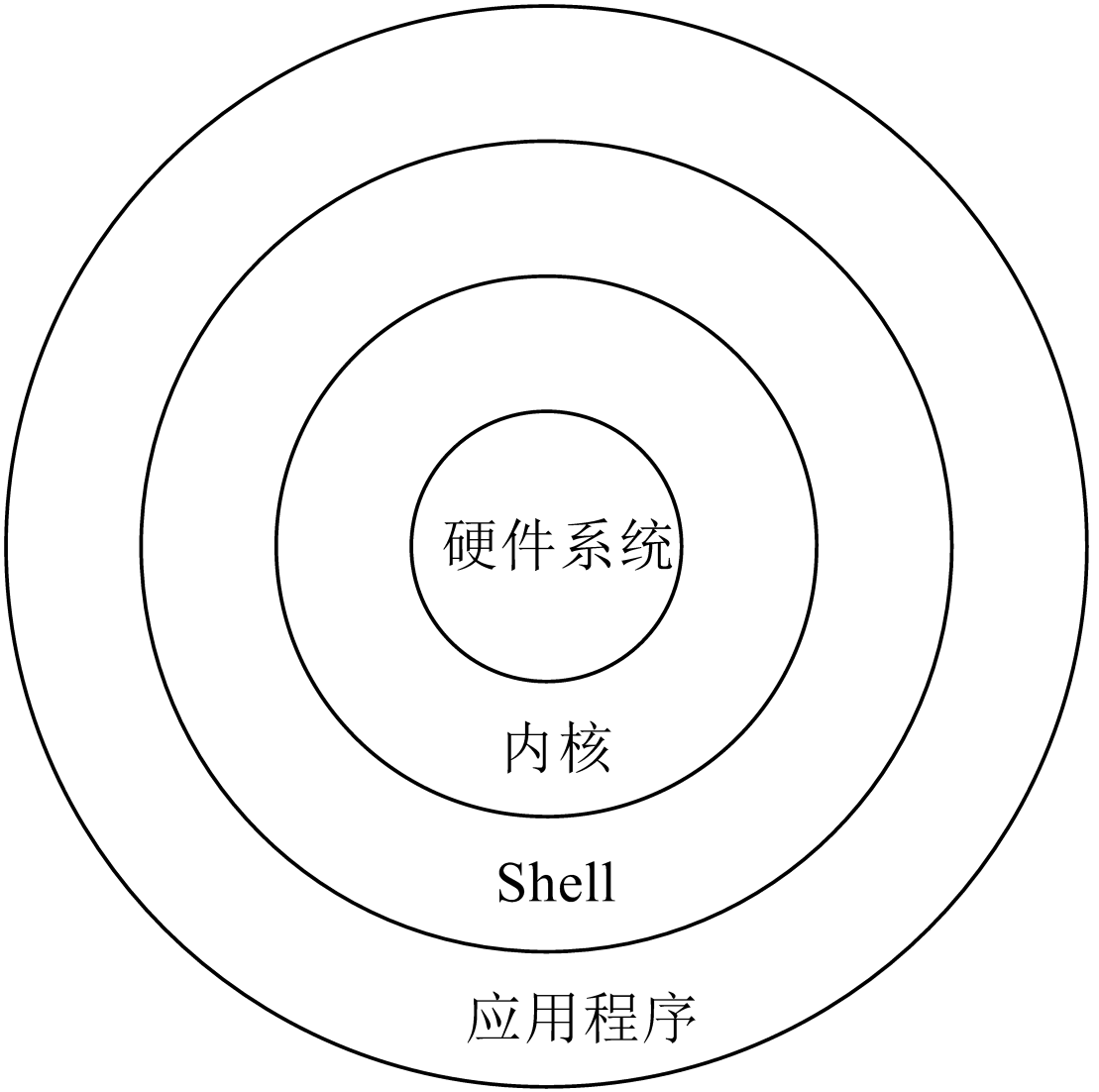
Linux Shell Scripts

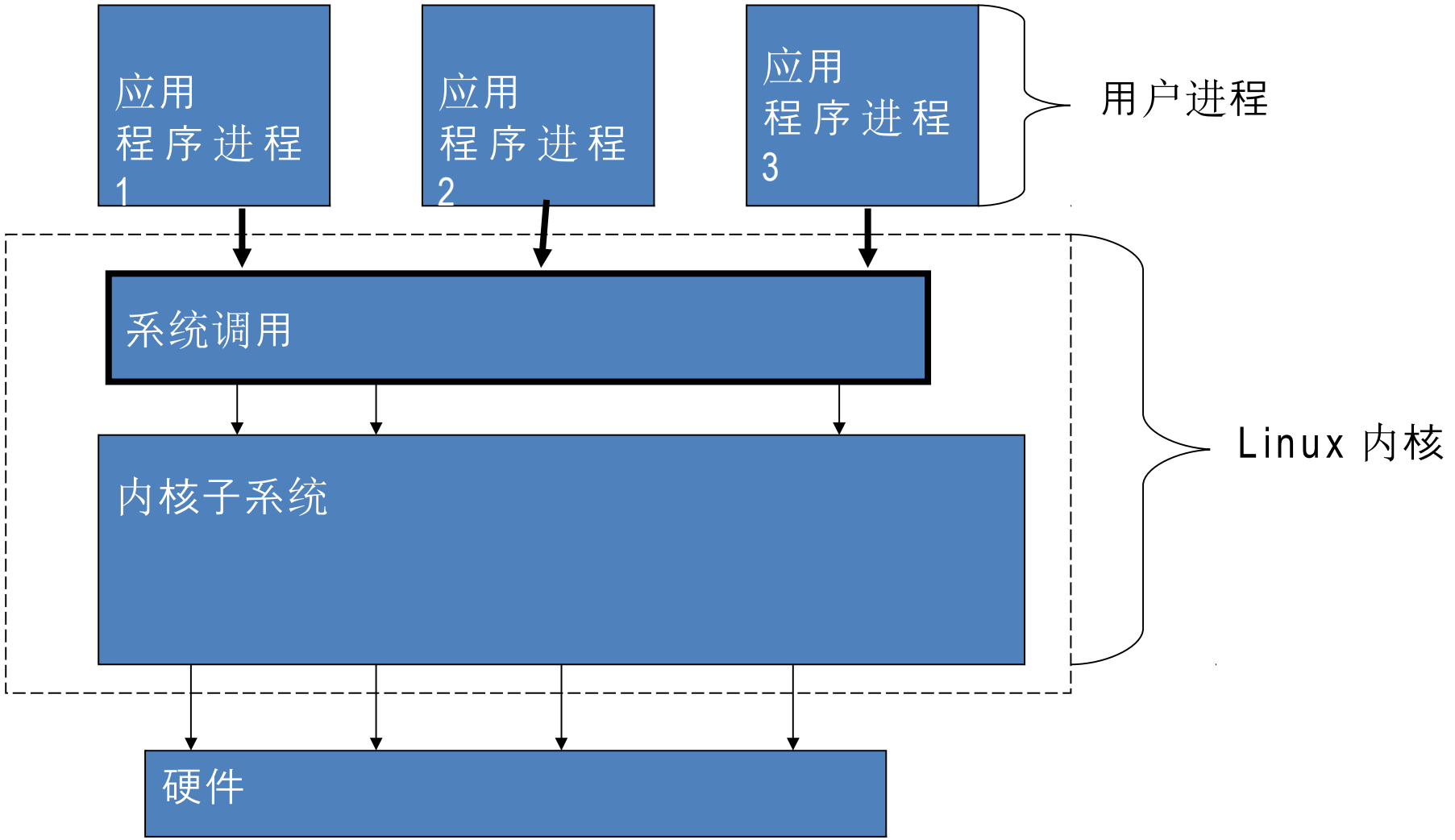
授课教师：李正华

Unix 操作系统的结构



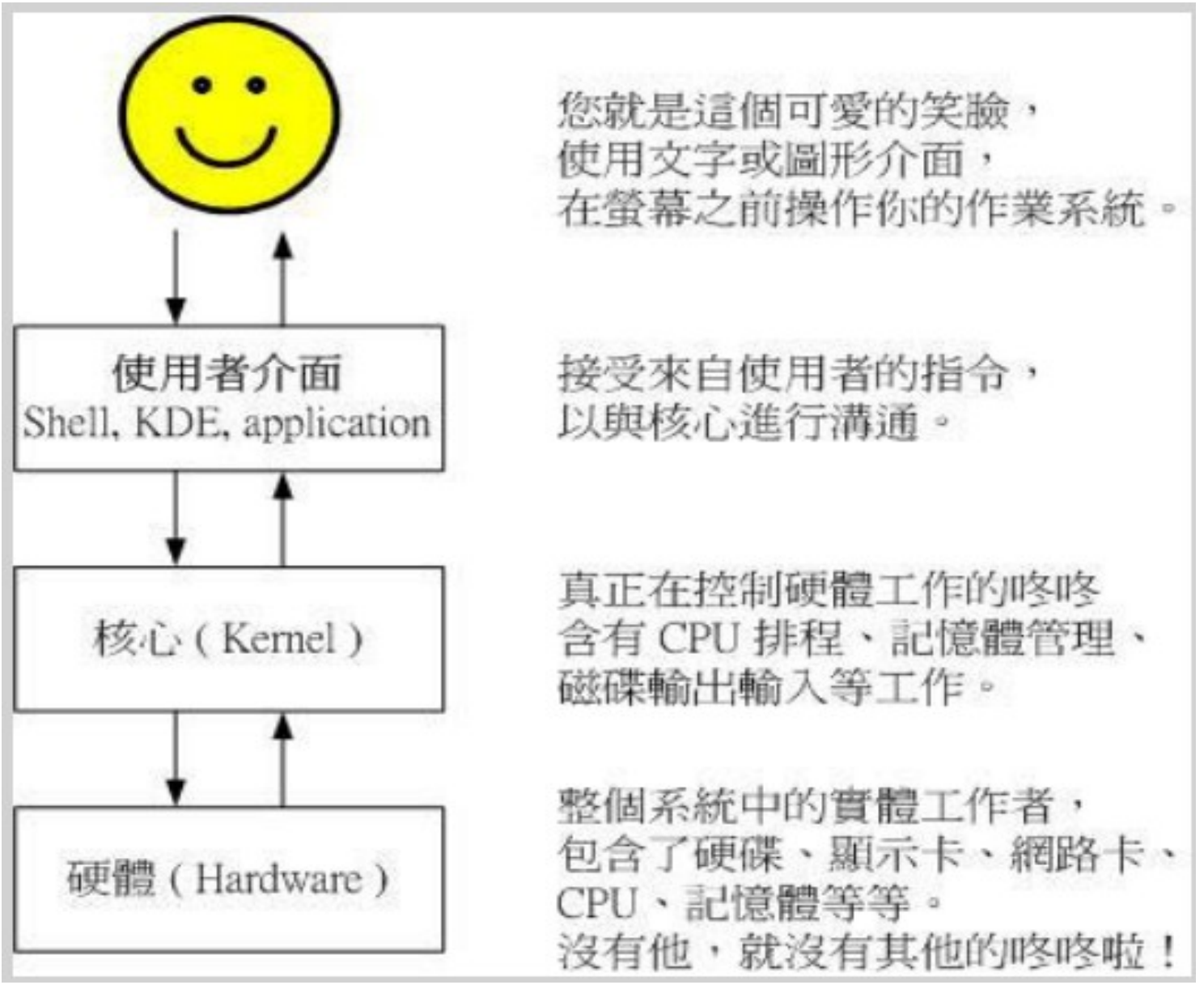
图片来自《unix 操作系统》清华大学

linux 操作系统的架构



图片来自赵有恩老师《Linux 操作系统》课件

硬件、核心与用户（鸟哥 .. ）

- 
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。
使用者介面
Shell, KDE, application
接受來自使用者的指令，
以與核心進行溝通。
核心（Kernel）
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。
硬體（Hardware）
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。
沒有他，就沒有其他的哆哆啦！

为什么要学习 Shell

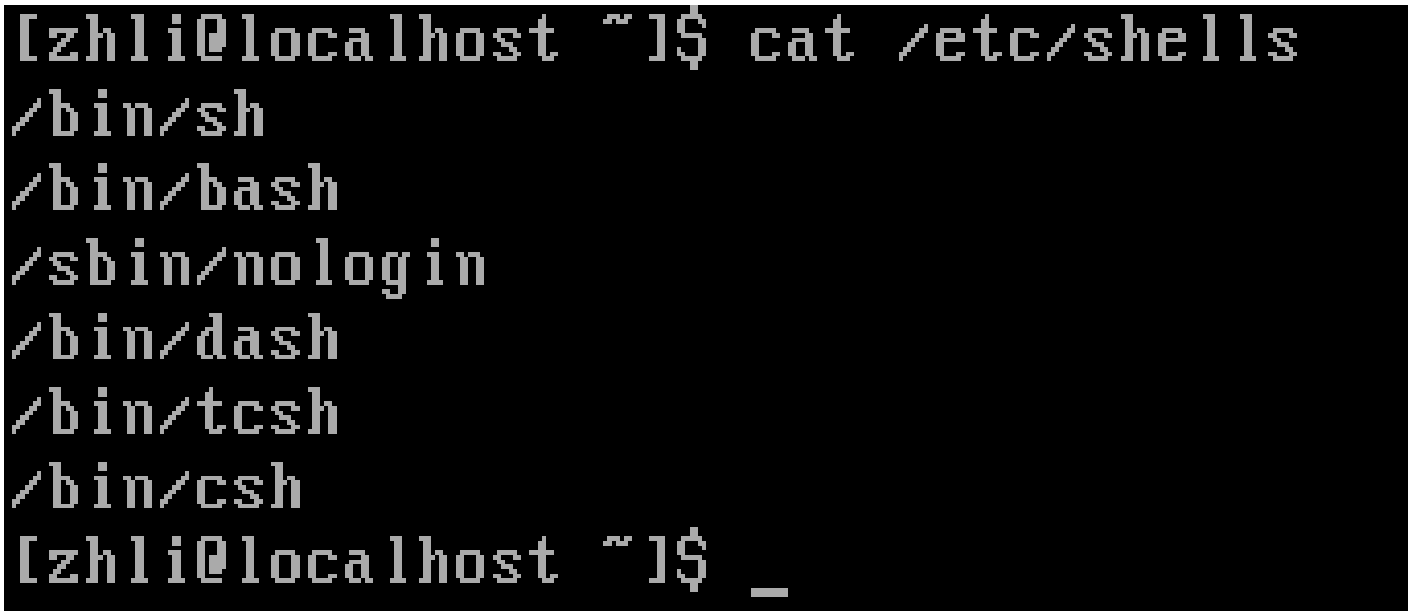
- 文字接口的 Shell ：大家都一样！
  - 无论什么 distribution ，都差不多
  - X Window 虽然方便，但不灵活，遇到问题时难以灵活的解决。
- 远程管理：文字界面就是比较快！
- Linux 的任督二脉： shell 是也！
  - 管理多台主机，利用 shell 脚本很高效！
- ...

Shell 有很多种

- /bin/sh
  - 第一个流行的 shell ，由 Steven Bourne 开发，称为 Bourne shell ，简称 sh
- /bin/bash
  - Linux 的标准 shell ，兼容 sh ，是 sh 的加强版
- /bin/csh
  - 语法类似 C 语言
- /bin/ksh
- ...

•如何查看系统可用的 shell

- cat /etc/shells



如何决定用户使用哪个 Shell ？

- cat /etc/passwd

```
[zhli@localhost ~]$ egrep 'zhli|root|nobody' /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
zhli:x:500:500:Zhenghua Li:/home/zhli:/bin/bash
[zhli@localhost ~]$ _
```

Bash Shell 的功能

- /bin/bash
- 命令记忆（ history ）： ~/.bash_history

```
  998  cd work/gdp-crf/src/
  999  vim Parser.cpp
 1000  vim Decoder_1o.cpp
 1001  ls
 1002  cat /etc/passwd
 1003  cat /etc/passwd | less
 1004  cat /etc/shells
 1005  egrep 'zhli|root|nobody' /etc/passwd
 1006  clear
 1007  history
[zhli@localhost ~]$ _
```

Bash Shell 的功能

- 命令和文档名称补全（ [tab] ）

Bash Shell 的功能

- 命令别名设定（ alias ）
  - alias lm= ‘ls -al'
-

```
[zhli@localhost work]$ cat ~/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
alias lm='ls -la'
[zhli@localhost work]$ lm
total 16
drwxrwxr-x.  4 zhli zhli 4096 Sep 16 16:50 .
drwx------. 36 zhli zhli 4096 Sep 23 19:38 ..
drwxrwxr-x.  4 zhli zhli 4096 Sep 16 16:50 data
drwxrwxrwx.  9 zhli zhli 4096 Sep 19 22:48 gdp-crf
[zhli@localhost work]$ _
```

Bash Shell 的功能

- 工作控制，前景背景控制（ job control ， foreground, background ）
- ctrl + c ：停掉当前程序
- 背景（后台）运行程序
  - tar -czvf xx.tar.gz *.dat &

Bash Shell 的功能

- 程序化脚本（ shell scripts ）
  - 让经常使用的命令集合或工作自动化
  - 高效的管理主机或服务器

Bash Shell 的功能

- 通配符（ wildcard ）
  - ls -l /user/bin/ex*

```
[zhli@localhost work]$ ls -l /usr/bin/ex*
lrwxrwxrwx. 1 root root      3 Sep 16 17:01 /usr/bin/ex -> vim
-rwxr-xr-x. 1 root root 184568 Aug 19 20:36 /usr/bin/execstack
-rwxr-xr-x. 1 root root  30504 May 23 19:00 /usr/bin/expand
-rwxr-xr-x. 1 root root 100224 May 23 19:00 /usr/bin/expr
-rwxr-xr-x. 1 root root 119592 Aug 17  2010 /usr/bin/expresskeys
```

Bash Shell 的功能

- 内建命令 type
  - 外部指令 vs. bash 内置的命令？

```
[zhli@localhost work]$ type date who  mv mkdir ls cd lm
date is /bin/date
who is /usr/bin/who
mv is /bin/mv
mkdir is /bin/mkdir
ls is hashed (/bin/ls)
cd is a shell builtin
lm is aliased to `ls -la'
```

Bash Shell 的功能

- 下达指令
  - ls, date, cd ...
  - 多行输入指令

```
[zhli@localhost ptb-data]$ cp dev.fltd-pos-0.01-dep-0.0001.conll06 \
> train.fltd-pos-0.01-dep-0.0001.conll06 \
> test.fltd-pos-0.01-dep-0.0001.conll06 \
> ../
```

变量

- int x = 3;
- int y = 10*x + 2;
- Linux 下的一些常用变量
  - PATH; HOME; SHELL

```
[zhli@localhost ptb-data]$ echo $HOME
/home/zhli
[zhli@localhost ptb-data]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/zhli/bin
[zhli@localhost ptb-data]$ echo $SHELL
/bin/bash
```

变量的读取（输出）

- echo $PATH
- echo ${PATH}

变量的定义和赋值

- myname=Zhenghua（中间不能有空格）
- x1= “i am $myname”
- x2='i am $myname'

```
[zhli@localhost ptb-data]$ echo $myname

[zhli@localhost ptb-data]$ myname=zhenghua
[zhli@localhost ptb-data]$ echo $myname
zhenghua
[zhli@localhost ptb-data]$ echo ${myname}
zhenghua
[zhli@localhost ptb-data]$ x1='i am $myname'
[zhli@localhost ptb-data]$ echo $x1
i am $myname
[zhli@localhost ptb-data]$ x2="i am $myname"
[zhli@localhost ptb-data]$ echo x2
x2
[zhli@localhost ptb-data]$ echo $x2
i am zhenghua
```

变量内容的扩展

- PATH="$PATH":/home/bin

```
[zhli@localhost ~]$ PATH=$PATH:/home/nobody
[zhli@localhost ~]$ echo PATH
PATH
[zhli@localhost ~]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/zhli/bin:/hom
e/nobody
[zhli@localhost ~]$ _
```

令变量在子 bash 中可见

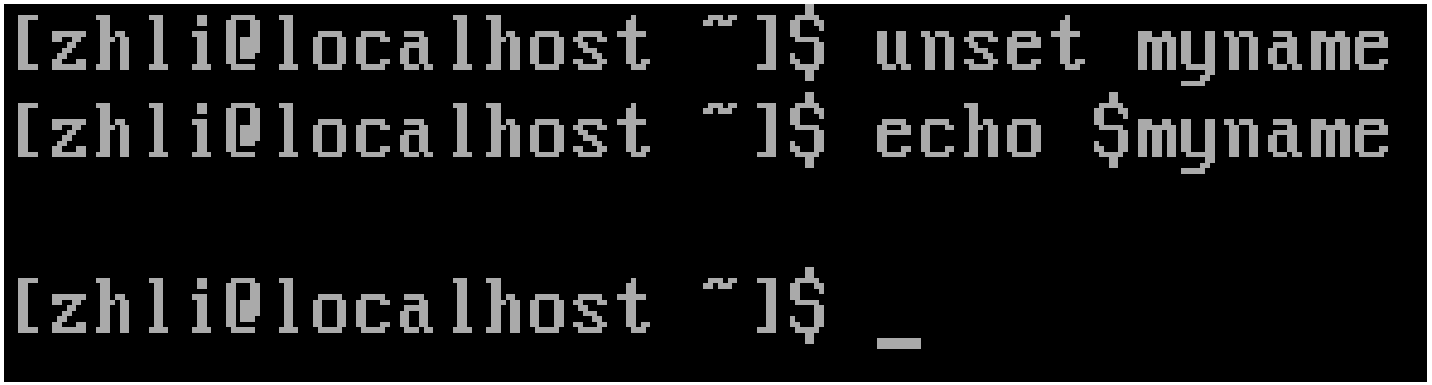
- export x2

```
[zhli@localhost ~]$ echo $x2
i am zhenghua
[zhli@localhost ~]$ bash
[zhli@localhost ~]$ echo x2
x2
[zhli@localhost ~]$ echo $x2

[zhli@localhost ~]$ exit
exit
[zhli@localhost ~]$ export x2
[zhli@localhost ~]$ bash
[zhli@localhost ~]$ echo $x2
i am zhenghua
[zhli@localhost ~]$ _
```

取消变量（删除）

- unset myname



系统环境变量的功能

- 家目录 : HOME
- shell 的选择： SHELL
- 执行文件的搜索： PATH
- 用户名字： USER

查看所有系统环境变量 env

```
HOSTNAME=localhost.localdomain
SHELL=/bin/bash
TERM=linux
XDG_SESSION_COOKIE=769970869e488d2f44a65264000000026-1379936351.731724-1739260654
HISTSIZE=1000
USER=zhli
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd
=40;33;01:or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;4
2:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:
*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:
*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.tbz=01;31:*.tbz2=01;31:*.bz=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01
;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bm
p=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:
*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=
01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.ogm=01;35:*
.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;
35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv
=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.e
mf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:
*.flac=01;36:*.mid=01;36:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=
01;36:*.ra=01;36:*.wav=01;36:*.axa=01;36:*.oga=01;36:*.spx=01;36:*.xspf=01;36:
x2=i am zhenghua
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/zhli/bin
:
```

查看所有系统环境变量 env

```
x2=i am zhenghua
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin
:/home/nobody
MAIL=/var/spool/mail/zhli
PWD=/home/zhli/work
LANG=en_US.UTF-8
HISTCONTROL=ignoredups
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpas
HOME=/home/zhli
SHLVL=2
LOGNAME=zhli
LESSOPEN=|/usr/bin/lesspipe.sh %s
G_BROKEN_FILENAMES=1
_=/bin/env
OLDPWD=/home/zhli
```

查看所有系统环境变量 set

```
PIPESTATUS=([0]="0")
PPID=2635
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/home/zhli
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:h
tor
SHLVL=1
SSH_ASKPASS=/usr/libexec/openssh/gnom
TERM=linux
UID=500
USER=zhli
XDG_SESSION_COOKIE=769970869e488d2f44
_=ls
colors=/etc/DIR_COLORS
x1='i am $myname'
x2='i am zhenghua'
__udisks ()
{
```

两个特殊的变量

- $ ：本 shell 的 PID （ process id ）
- ？：上一个指令的返回值（ 0 vs. other ）

```
[zhli@localhost ~]$ echo $$
2641
[zhli@localhost ~]$ bash
[zhli@localhost ~]$ echo $$
3412
[zhli@localhost ~]$ exit
exit
[zhli@localhost ~]$ echo $$
2641
[zhli@localhost ~]$ echo $?
0
[zhli@localhost ~]$ myname = 1
-bash: myname: command not found
[zhli@localhost ~]$ echo $?
127
[zhli@localhost ~]$ echo $?
0
[zhli@localhost ~]$ _
```

自定义变量变为环境变量 export

- 父进程（程序， shell ）
- 子进程（程序， shell ）
  - 儿子会继承父亲的环境变量
  - 不继承自定义变量

影响显示结果的语系变量 locale

```
[zhli@localhost ~]$ locale -a | tail -10
zh_SG.gb2312
zh_SG.gbk
zh_SG.utf8
zh_TW
zh_TW.big5
zh_TW.euctw
zh_TW.utf8
zu_ZA
zu_ZA.iso88591
zu_ZA.utf8
[zhli@localhost ~]$ _
```

影响显示结果的语系变量 locale

```
[zhli@localhost ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
[zhli@localhost ~]$ _
```

变量键盘读取

- read

```
[zhli@localhost ~]$ read hello
hello world
[zhli@localhost ~]$ echo $hello
hello world
```

变量键盘读取、数组与宣告

- read
- array
- declare
-

```
[root@www ~]# declare [-aixr] variable
选项与参数：
-a   ：将后面名为 variable 的变量定义成为数组 (array) 类型
-i   ：将后面名为 variable 的变量定义成为整数数字 (integer) 类型
-x   ：用法与 export 一样，就是将后面的 variable 变成环境变量；
-r   ：将变量设定成为 readonly 类型，该变量不可被更改内容，也不能 unset

范例一：让变量 sum 进行 100+300+50 的加总结果
```

变量宣告

- declare

```
[root@www ~]# declare [-aixr] variable
选项与参数：
-a   ：将后面名为 variable 的变量定义成为数组 (array) 类型
-i   ：将后面名为 variable 的变量定义成为整数数字 (integer) 类型
-x   ：用法与 export 一样，就是将后面的 variable 变成环境变量；
-r   ：将变量设定成为 readonly 类型，该变量不可被更改内容，也不能 unset
```

```
[root@www ~]# sum=100+300+50
[root@www ~]# echo $sum
100+300+50  <==咦！怎么没有帮我计算加总？因为这是文字型态的变量属性啊！
[root@www ~]# declare -i sum=100+300+50
[root@www ~]# echo $sum
450         <==瞧乎？？
```

变量数组

- array

```
[zhli@localhost ~]$ val[1]='hello'
[zhli@localhost ~]$ val[2]='world'
[zhli@localhost ~]$ val[3]='!'
[zhli@localhost ~]$ echo "$val[1] $val[2]$val[3]"
[1] [2][3]
[zhli@localhost ~]$ echo "${val[1]} ${val[2]}${val[3]}"
hello world!
```

变量内容的删除与取代

- ${val#keyword} ：从头删除（最短）
- ${val##keyword} ：从头删除（最长）
- ${val%keyword} ：从尾删除（最短）
- ${val%%keyword} ：从尾删除（最长）
- ${val/oldstr/newstr} ：取代第一个
- ${val//oldstr/newstr} ：取代所有
-

变量内容的删除与取代

- ${val#keyword} ：若变量内容从头开始的数据符合“ keyword” ，则将符合的最短数据删除

```
范例一：先让小写的 path 自定义变量设定的与 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
```

变量内容的删除与取代

```
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！
```

变量内容的删除与取代

```
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！
```

变量内容的删除与取代

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！
```

变量内容的删除与取代

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！
```

变量的测试与内容替换

- 判断变量是否存在
  - 若存在，则使用既有的设定
  - 若不存在，则给予一个常用的设定

```
[zhli@localhost ~]$ unset val
[zhli@localhost ~]$ echo $val

[zhli@localhost ~]$ val=${val-hello}
[zhli@localhost ~]$ echo $val
hello
[zhli@localhost ~]$ val=${val-hello2}
[zhli@localhost ~]$ echo $val
hello
```

变量的测试与内容替换

- 如果变量为空字符串？

```
[zhli@localhost ~]$ val=''
[zhli@localhost ~]$ echo $val

[zhli@localhost ~]$ val=${val:-hello}
[zhli@localhost ~]$ echo $val
hello
[zhli@localhost ~]$ val=${val:-helloZ}
[zhli@localhost ~]$ echo $val
hello
```

变量的测试与内容替换

变量设定方式	str 没有设定	str 为空字符串	str 已设定非为空字符串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=	str 不变 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不变 var=$str
var=${str?expr}	expr 输出至 stderr	var=	var=$str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=$str

login shell 配置文件读入过程

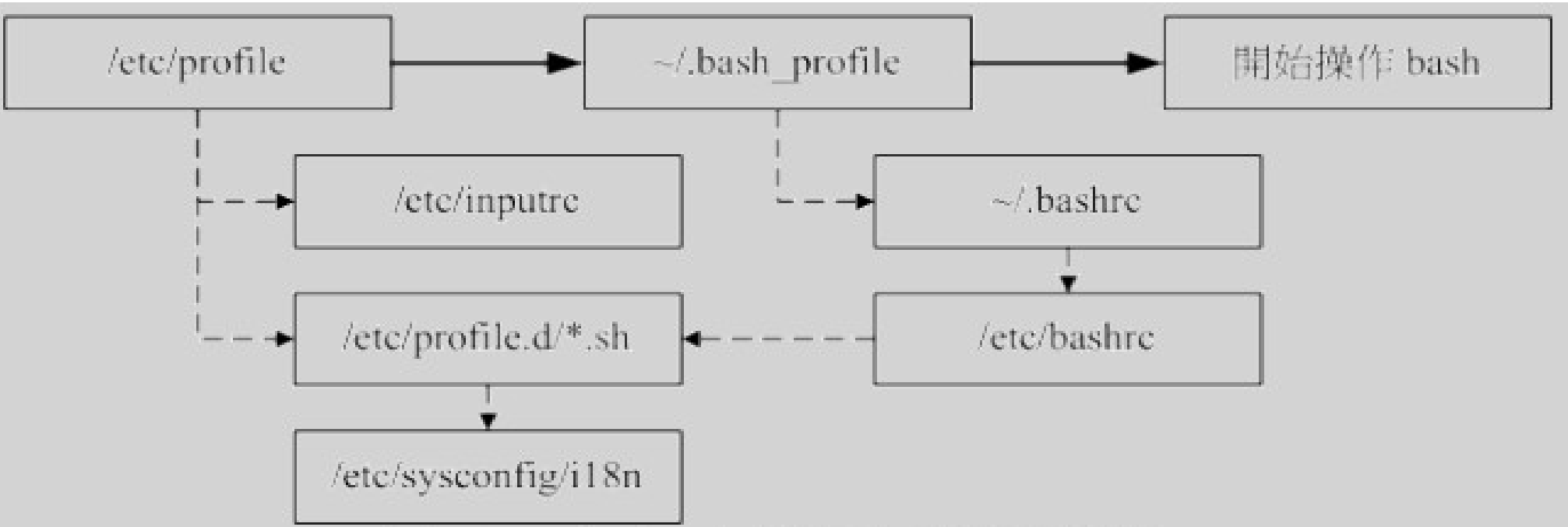


图 4.3.1、login shell 的配置文件读取流程

source 读入环境配置文件

- 一个人的工作环境分为几种情况，需要不同的方案。
- 无需注销再登陆
- source ~/.bashrc1
- source ~/.bashrc2
- ..

命令执行的判断依据

- ； // 一次顺序执行多个指令

```
[root@www ~]# sync; sync; shutdown -h now
```

命令执行的判断依据

- 指令相关性：根据前一个指令的结果（ $? ）决定下一个指令的执行

指令下达情况	说明
cmd1 && cmd2	1. 若 cmd1 执行完毕且正确执行($?=0)，则开始执行 cmd2。 2. 若 cmd1 执行完毕且为错误 ($?≠0)，则 cmd2 不执行。
cmd1 || cmd2	1. 若 cmd1 执行完毕且正确执行($?=0)，则 cmd2 不执行。 2. 若 cmd1 执行完毕且为错误 ($?≠0)，则开始执行 cmd2。

命令相关性的例子 &&

范例一：使用 ls 查阅目录 /tmp/abc 是否存在，若存在则用 touch 建立 /tmp/abc/hehe

```
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
ls: /tmp/abc: No such file or directory
```

ls 很干脆的说明找不到该目录，但并没有 touch 的错误，表示 touch 并没有执行

```
[root@www ~]# mkdir /tmp/abc
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
[root@www ~]# ll /tmp/abc
-rw-r--r-- 1 root root 0 Feb  7 12:43 hehe
```

命令相关性的例子 ||

范例二：测试 /tmp/abc 是否存在，若不存在则予以建立，若存在就不作任何事情

```
[root@www ~]# rm -r /tmp/abc              <==先删除此目录以方便测试
[root@www ~]# ls /tmp/abc || mkdir /tmp/abc
ls: /tmp/abc: No such file or directory <==真的不存在喔！
[root@www ~]# ll /tmp/abc
total 0                                   <==结果出现了！有进行 mkdir
```

参数替换命令 xargs

- 读入 stdin 的数据，以空格或换行符，分割为 arguments
- 很多命令不支持管线命令，可以用 xargs 使得这些命令引用 stdin 。

参数替换命令 xargs

```
范例五：找出 /sbin 底下具有特殊权限的档名，并使用 ls -l 列出详细属性
[root@www ~]# find /sbin -perm +7000 | ls -l
# 结果竟然仅有列出 root 所在目录下的档案！这不是我们要的！
# 因为 ll (ls) 并不是管线命令的原因啊！

[root@www ~]# find /sbin -perm +7000 | xargs ls -l
-rwsr-xr-x 1 root root 70420 May 25  2008 /sbin/mount.nfs
-rwsr-xr-x 1 root root 70424 May 25  2008 /sbin/mount.nfs4
-rwxr-sr-x 1 root root  5920 Jun 15  2008 /sbin/netreport
....(底下省略)....
```