



Ch2 Shell Programming

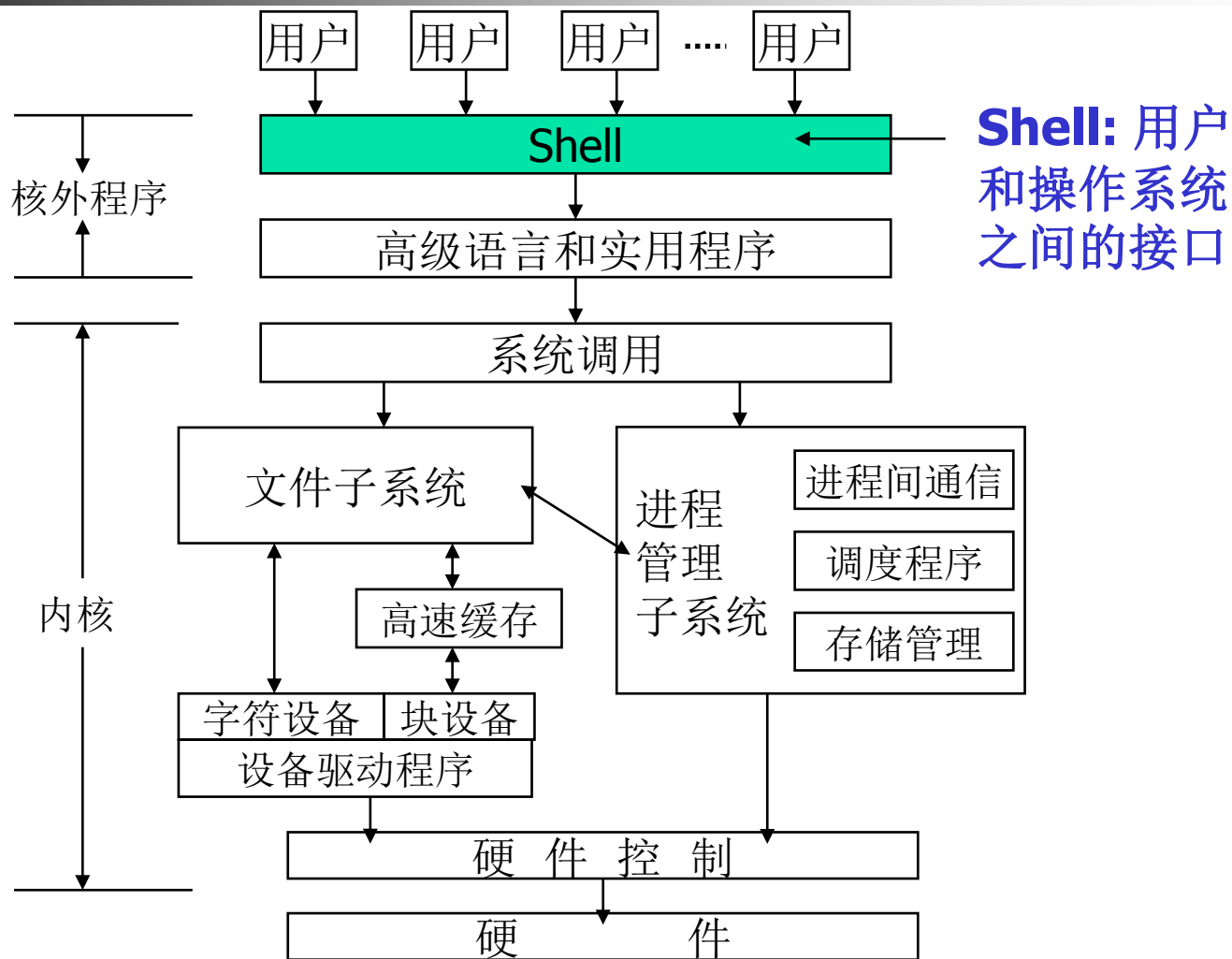
Enyi Tang
Software Institute, Nanjing
University



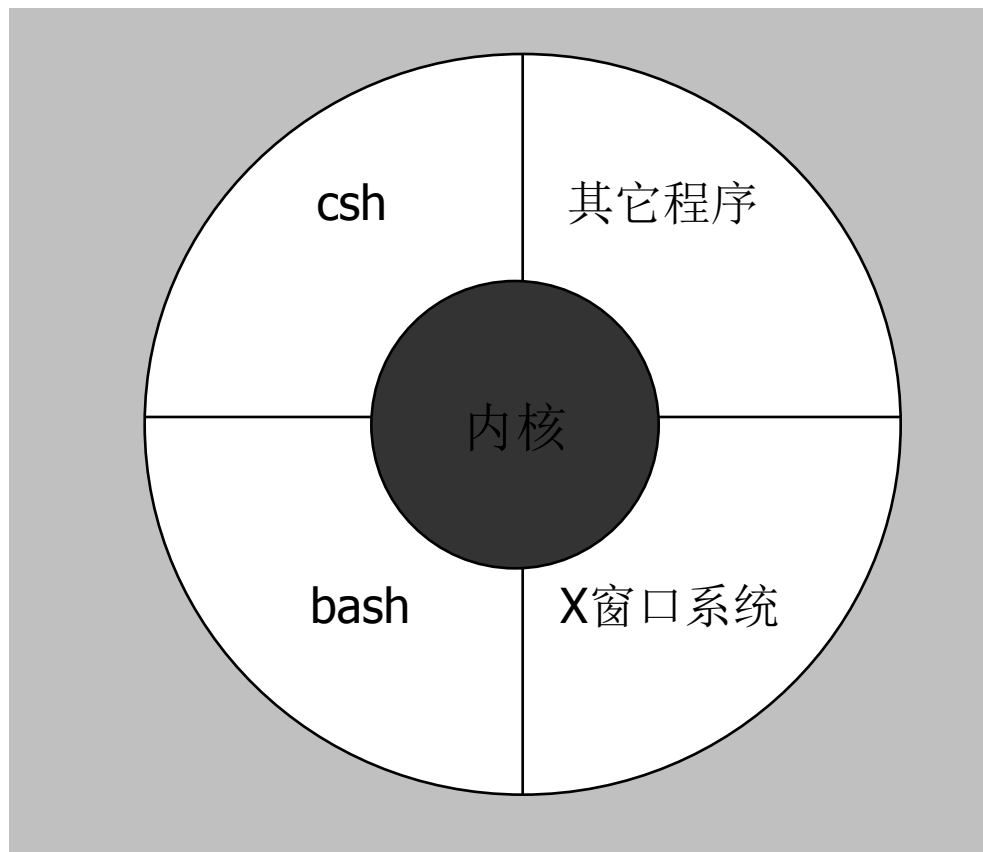
What is Shell?

- Shell:
 - A command interpreter and programming environment
- 用户和操作系统之间的接口
- 作为核外程序而存在

Shell: 用户和操作系统之间的接口



Shell: 作为核外程序而存在





各种不同的Shell

shell名称	描述	位置
ash	一个小的shell	/bin/ash
ash.static	一个不依靠软件库的ash版本	/bin/ash.static
bsh	ash的一个符号链接	/bin/bsh
bash	“Bourne Again Shell”。Linux中的主角，来自GNU项目	/bin/bash
sh	bash的一个符号链接	/bin/sh
csh	C shell, tcsh的一个符号链接	/bin/csh
tcsh	和csh兼容的shell	/bin/tcsh
ksh	Korn Shell	/bin/ksh



Shell 的双重角色

- 命令解释程序
 - Linux的开机启动过程；进程树
 - Shell的工作步骤
 - 打印提示符；得到命令行；解析命令；查找文件；准备参数；执行命令
- 独立的程序设计语言解释器
 - KISS (Keep It Small and Stupid)
 - Reusable tools
 - Redirection and pipe



UNIX's Philosophy (Examples)

- Redirection
 - Use “echo” to create a file?
- Pipe
 - Get the count of files in a directory?
 - Show subdirectories and only subdirectories?
 - `ar t /usr/lib/libc.a | grep printf | pr -4 -t (?)`



脚本是能在命令行直接输入的

- 但仅会执行一次

```
$ for file in *  
> do  
> if grep -l POSIX $file  
> then  
> more $file  
> fi  
> done  
posix  
This is a file with POSIX in it - treat it well  
$
```




编写脚本文件

- 脚本文件
 - 注释
 - 退出码(exit code)
 - Example

```
#!/bin/bash

# Here is comments

for file in *; do
    if grep -l POSIX $file; then
        more $file
    fi
done

exit 0
```



执行脚本文件

- 方法1:
 - \$ sh script_file
- 方法2:
 - chmod +x script_file (chown, chgrp optionally)
 - ./script_file
- 方法3:
 - source script_file, or
 - . script_file



用户环境

- **.bash_profile, .bash_logout, .bashrc files**
 - **.bash_profile**: 用户登录时被读取，其中包含的命令被bash执行
 - **.bashrc**: 启动一个新的shell时读取并执行
 - **.bash_logout**: 登录退出时读取执行
- **Alias**
 - **alias/unalias command**
- **环境变量**
 - **export command**
 - **export, env & set command**



变量

- 用户变量
- 环境变量
- 参数变量和内部变量



用户变量

- 用户变量：
 - 用户在shell脚本里定义的变量
- 变量的赋值和使用
 - `var=value`
 - `echo $var`
- `read`命令
 - 用法: `read var` 或 `read`
 - `REPLY` variable
- 引号的用法
 - 双引号, 单引号
 - 转义符 “\”



Read的用法

```
#!/bin/bash
echo -n "Enter your name:"          #参数-n的作用是不换行，echo默认是换行
read name                          #从键盘输入
echo "hello $name, welcome to my program"
exit 0                             #退出shell程序。
```

```
read -p "Enter your name:" name    #-p参数，允许在read命令行中直接指定一个提示
```



Read 使用

```
read -p "Enter a number"  
echo $REPLY
```

```
#!/bin/bash  
if read -t 5 -p "please enter your name:" name  
then  
    echo "hello $name, welcome to my script"  
else  
    echo "sorry,too slow"  
fi  
exit 0
```



Read 使用

```
#!/bin/bash
read -nl -p "Do you want to continue [Y/N]? " answer
case $answer in
Y|y)
    echo "fine ,continue";;
N|n)
    echo "ok,good bye";;
*)
    echo "error choice";;
esac
exit 0
```




Read 使用

```
#!/bin/bash
read -s -p "Enter your password: " pass
echo "your password is $pass"
exit 0
```

```
#!/bin/bash
count=1
cat dat| while read line
do
    echo "$count:$line"
    count=$((count + 1))
done
exit 0
```



引号的用法

- 单引号内的所有字符都保持它本身字符的意思，而不会被**bash**进行解释，例如，**\$**就是**\$**本身而不再是**bash**的变量引用符；****就是****本身而不再是**bash**的转义字符。
。
- 除了**\$**、**`**（不是单引号）和****外，双引号内的所有字符将保持字符本身的含义而不被**bash**解释。



环境变量

- 环境变量：
 - Shell环境提供的变量。通常使用大写字母做名字

环境变量	说明
\$HOME	当前用户的登陆目录
\$PATH	以冒号分隔的用来搜索命令的目录清单
\$PS1	命令行提示符，通常是"\$"字符
\$PS2	辅助提示符，用来提示后续输入，通常是">"字符
\$IFS	输入区分隔符。当shell读取输入数据时会把一组字符看成是单词之间的分隔符，通常是空格、制表符、换行符等。



参数变量和内部变量

- 参数变量和内部变量：
 - 调用脚本程序时如果带有参数，对应的参数和额外产生的一些变量。

环境变量	说明
<hr/>	
\$#	传递到脚本程序的参数个数
\$0	脚本程序的名字
\$1, \$2, ...	脚本程序的参数
\$*	一个全体参数组成的清单，它是一个独立的变量，各个参数之间用环境变量IFS中的第一个字符分隔开
\$@	“\$*”的一种变体，它不使用IFS环境变量。



条件测试

- 退出码
- **test**命令
 - test expression 或 [expression]
- **test**命令支持的条件测试
 - 字符串比较
 - 算术比较
 - 与文件有关的条件测试
 - 逻辑操作



字符串比较

字符串比较

结果

`str1 = str2`

两个字符串相同则结果为真

`str1!=str2`

两个字符串不相同则结果为真

`-z str`

字符串为空则结果为真

`-n str`

字符串不为空则结果为真



算术比较

算术比较

结果

`expr1 -eq expr2`

两个表达式相等则结果为真

`expr1 -ne expr2`

两个表达式不等则结果为真

`expr1 -gt expr2`

`expr1` 大于 `expr2` 则结果为真

`expr1 -ge expr2`

`expr1` 大于或等于 `expr2` 则结果为真

`expr1 -lt expr2`

`expr1` 小于 `expr2` 则结果为真

`expr1 -le expr2`

`expr1` 小于或等于 `expr2` 则结果为真



与文件有关的条件测试

文件条件测试

结果

-e file	文件存在则结果为真
-d file	文件是一个子目录则结果为真
-f file	文件是一个普通文件则结果为真
-s file	文件的长度不为零则结果为真
-r file	文件可读则结果为真
-w file	文件可写则结果为真
-x file	文件可执行则结果为真



逻辑操作

逻辑操作	结果
! expr	逻辑表达式求反
expr1 -a expr2	两个逻辑表达式 “And”（“与”）
expr1 -o expr2	两个逻辑表达式 “Or”（“或”）



条件语句

- if语句
- case语句



if语句(1)

- 形式

```
if [ expression ]  
then  
    statements  
elif [ expression ]  
then  
    statements  
elif ...  
else  
    statements  
fi
```

- 紧凑形式

- ; (同一行上多个命令的分隔符)



if语句(2)

- 例1(.bash_profile文件中)

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```
- 例2

```
#!/bin/sh
echo "Is this morning? Please answer yes or no."
read answer
if [ "$answer" = "yes" ]; then
    echo "Good morning"
elif [ "$answer" = "no" ]; then
    echo "Good afternoon"
else
    echo "Sorry, $answer not recognized. Enter yes or no"
    exit 1
fi
exit 0
```



case语句(1)

- 形式

```
case str in
```

```
    str1 | str2) statements;;
```

```
    str3 | str4) statements;;
```

```
    *) statements;;
```

```
esac
```



case语句(2)

- Example

```
#!/bin/sh
```

```
echo "Is this morning? Please answer yes or no."
```

```
read answer
```

```
case "$answer" in
```

```
    yes | y | Yes | YES) echo "Good morning!" ;;
```

```
    no | n | No | NO) echo "Good afternoon!" ;;
```

```
    *) echo "Sorry, answer not recognized." ;;
```

```
esac
```

```
exit 0
```



重复语句

- for语句
- while语句
- until语句
- select语句



for语句(1)

- 形式

```
for var in list
```

```
do
```

```
    statements
```

```
done
```

- 适用于对一系列字符串循环处理



for语句(2)

- Example

```
#!/bin/sh
```

```
for file in $(ls f*.sh); do
```

```
    lpr $file
```

```
done
```

```
exit 0
```



while语句(1)

- 形式

while condition

do

statements

done



while语句(2)

- Example

```
quit=n
while [ "$quit" != "y" ]; do
    read menu_choice
    case "$menu_choice" in
        a) do_something;;
        b) do_anotherthing;;
        ...
        q|Q) quit=y;;
        *) echo "Sorry, choice not recognized.";;
    esac
done
```



while语句(3)

```
a=0
while [ "$a" -le "$LIMIT" ]
do
    a=$((a+1))
    if [ "$a" -gt 2 ]
    then
        break # Skip entire rest of loop.
    fi
    echo -n "$a "
done
```



until语句

- 形式

until condition

do

statements

done

- Not recommended (while statement is preferred)



select语句(1)

- 形式

```
select item in itemlist  
do  
    statements  
done
```

- 作用

- 生成菜单列表



select语句(2)

- 例(一个简单的菜单选择程序)

```
#!/bin/sh
```

```
clear
```

```
select item in Continue Finish
```

```
do
```

```
    case "$item" in
```

```
        Continue) ;;
```

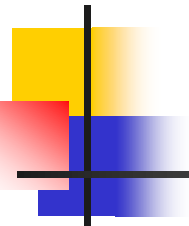
```
        Finish) break ;;
```

```
        *) echo "Wrong choice! Please select again!" ;;
```

```
    esac
```

```
done
```

- Homework: 用while语句模拟



```
#!/bin/bash
clear
select item in Continue Finish
do
    echo $item
    case "$item" in
        Continue) echo "Hit Continue" ;;
        Finish) echo "Hit Finish, now quit"; break ;;
        *) echo "Wrong choice! Please select again!" ;;
    esac
done
```




运行结果

```
1) Continue
2) Finish
#? 4

"Wrong choice! Please select again!"
#? 1
Continue
Hit Continue
#? 2
Finish
Hit Finish, now quit
brushington@vmware:~/Desktop$
```



命令表和语句块

- 命令表 (命令组合)
- 语句块



命令表

- 命令组合

- 分号串联

- `command1 ; command2 ; ...`

- 条件组合

- AND命令表

- 格式: `statement1 && statement2 && statement3 && ...`

- OR命令表

- 格式: `statement1 || statement2 || statement3 || ...`



命令表

- 命令组合

- 分号串联

- `command1 ; command2 ; ...`

- 条件组合

- AND命令表

- 格式: `statement1 && statement2 && statement3 && ...`

- OR命令表

- 格式: `statement1 || statement2 || statement3 || ...`



语句块

- 形式

{

statement1

statement2

...

}

或 { statement1; statement2 ; ... ; }



函数

- 形式

```
func()  
{  
    statements  
}
```
- 局部变量
 - local关键字
- 函数的调用

```
func para1 para2 ...
```
- 返回值
 - return



函数的例子(1): 定义

```
yesno()
{
    msg="$1"
    def="$2"
    while true; do
        echo " "
        echo "$msg"
        read answer
        if [ -n "$answer" ]; then
            [REDACTED]
        else
            return $def
        fi
    done
}
```

```
case "$answer" in
    y|Y|yes|YES)
        return 0
        ;;
    n|N|no|NO)
        return 1
        ;;
    *)
        echo " "
        echo "ERROR: Invalid response,
expected \"yes\" or \"no\"."
        continue
        ;;
esac
```



函数的例子(2): 使用

- 调用函数**yesno**

```
if yesno "Continue installation? [n]" 1 ; then
    :
else
    exit 1
fi
```




其它

- 杂项命令
 - break, continue, exit, return, export, set, unset, trap, “:”, “.”, ...
- 捕获命令输出
- 算术扩展
- 参数扩展
- 即时文档



杂项命令

- **break**: 从for/while/until循环退出
- **continue**: 跳到下一个循环继续执行
- **exit n**: 以退出码“n”退出脚本运行
- **return**: 函数返回
- **export**: 将变量导出到shell, 使之成为shell的环境变量
- **set**: 为shell设置参数变量
- **unset**: 从环境中删除变量或函数
- **trap**: 指定在收到操作系统信号后执行的动作
- **“:”(冒号命令)**: 空命令
- **“.”(句点命令)或source**: 在当前shell中执行命令



捕获命令输出

- 语法

- `$(command)`
- ``command``

- 举例

```
#!/bin/sh
```

```
echo "The current directory is $PWD"
```

```
echo "The current directory is $(pwd)"
```

```
exit 0
```



算术扩展

- `$((...))`扩展

```
#!/bin/sh
```

```
x=0  
while [ "$x" -ne 10 ]; do  
    echo $x  
    x=$((x+1))  
done  
  
exit 0
```



参数扩展

- 问题:
 - 批处理 1_tmp, 2_tmp, ...

- 方法

```
#!/bin/sh
```

```
i=0
```

```
while [ "$i" -ne 10 ]; do
```

```
    touch "${i}_tmp"
```

```
    i=$((i+1))
```

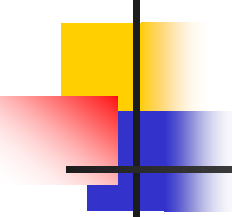
```
done
```

```
exit 0
```



参数扩展更复杂的形式

Parameter Expansion	Description
<code>\${param:-default}</code>	If param is null, then set it to the value of default.
<code>\${#param}</code>	Gives the length of param
<code>\${param%word}</code>	From the end, removes the smallest part of param that matches word and returns the rest
<code>\${param%%word}</code>	From the end, removes the longest part of param that matches word and returns the rest
<code>\${param#word}</code>	From the beginning, removes the smallest part of param that matches word and returns the rest
<code>\${param##word}</code>	From the beginning, removes the longest part of param that matches word and returns the rest



<code>\${param:-default}</code>	如果param为空，就把它设置为default的值
<code>\${#param}</code>	给出param的长度
<code>\${param%word}</code>	从param的尾部开始删除与word匹配的最小部分，然后返回剩余部分
<code>\${param%%word}</code>	从param的尾部开始删除与word匹配的最长部分，然后返回剩余部分
<code>\${param#word}</code>	从param的头部开始删除与word匹配的最小部分，然后返回剩余部分
<code>\${param##word}</code>	从param的头部开始删除与word匹配的最长部分，然后返回剩余部分



即时文档

- 在shell脚本中向一条命令传送输入数据

- Example

```
#!/bin/bash
```

```
cat >> file.txt << !CATINPUT!
```

```
Hello, this is a here document.
```

```
!CATINPUT!
```