




LINUX是什麼

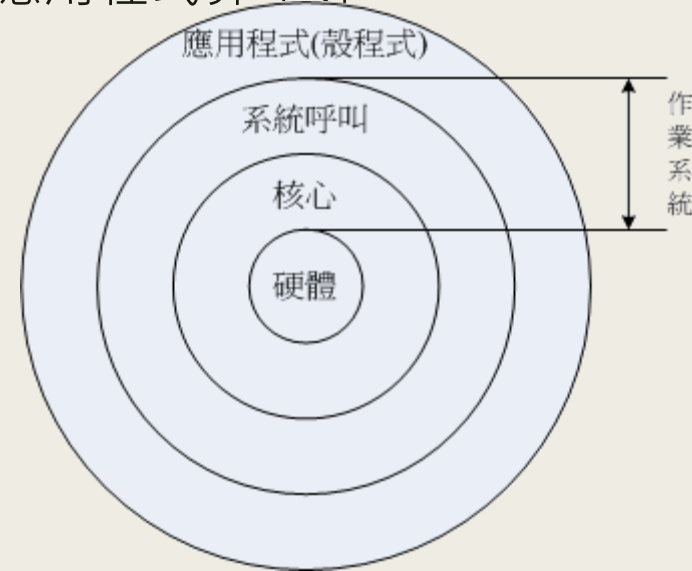
系統程式-期中報告
110710533 資工二 施泓宇



- 1.1 什麼是Linux
 - 1.1.1 Linux是什麼?作業系統?應用程式?-----03
 - 1.1.2 Linux的應用-----04
 - 1.1.3 Linux歷史-----05
- 1.2 Linux程序設計之shell程序設計
 - 1.2.1 前言
 - 1.2.2 重定向
 - 1.2.3 管道與通配符
 - 1.2.4 創建腳本
 - 1.2.5 if語句
 - 1.2.6 for語句
 - 1.2.7 while語句和until語句
 - 1.2.8 case語句
 - 1.2.9 函數

1.1.1 Linux是什麼？作業系統？應用程式？

- 電腦主機是由一堆硬體所組成的，為了有效率的控制這些硬體資源，於是乎就有作業系統的產生了。作業系統除了有效率的控制這些硬體資源的分配，並提供電腦運作所需要的功能(如網路功能)之外，為了要提供程式設計師更容易開發軟體的環境，所以作業系統也會提供一整組系統呼叫介面來給軟體設計師開發用！所以Linux就是一個作業系統！
- 如同下圖所示，Linux就是核心與系統呼叫介面那兩層。至於應用程式算不算Linux呢？當然不算啦！這點要特別注意喔！



1.1.2 Linux的應用

- 由上圖中我們可以看到其實核心與硬體的關係非常的強烈。早期的Linux是針對386來開發的，由於Linux只是一套作業系統並不含有其他的應用程式，因此很多工程師在下載了Linux核心並且實際安裝之後，就只能看著電腦開始運作了！接下來這些高級工程師為了自己的需求，再在Linux上面安裝他們所需要的軟體就是了。
- 由於不同的硬體他的功能函數並不相同，例如IBM的Power CPU與Intel的x86架構就是不一樣！所以同一套作業系統是無法在不同的硬體平台上面運作的！舉例來說，如果你想要讓x86上面跑的那套作業系統也能夠在Power CPU上運作時，就得要將該作業系統進行修改才行。如果能夠參考硬體的功能函數並據以修改你的作業系統程式碼，那經過改版後的作業系統就能夠在另一個硬體平台上面運作了。這個動作我們通常就稱為『軟體移植』了！
- EX: Windows作業系統在蘋果公司的麥金塔電腦(MAC)上面安裝與運作

1.1.3 Linux歷史

- Linux的前身為Unix，早在Linux出現之前的二十年(大約在1970 年代)，Unix就是一個相當穩定而成熟的作業系統了！
- Linux的核心是由Linus Torvalds在1991年的時候給他開發出來的， 並且丟到網路上供大家下載，後來大家覺得這個小東西(Linux Kernel)相當的小而精巧， 所以慢慢的就有相當多的朋友投入這個小東西的研究領域裡面去了！
- UNIX作業系統（英語：UNIX），是美國AT&T公司貝爾實驗室於1969年完成的作業系統。最早由肯·湯普遜（Ken Thompson），丹尼斯·里奇（Dennis Ritchie），道格拉斯·麥克羅伊（Douglas McIlroy），和喬伊·歐桑納於1969年在AT&T貝爾實驗室開發。於1971年首次發布，最初是完全用組合語言編寫。後來，在1973年用一個重要的開拓性的方法，Unix被丹尼斯·里奇用程式語言C（核心和I/O例外）重新編寫^[11]。高階語言編寫的作業系統具有更佳的相容性，能更容易地移植到不同的電腦平台。

1.2.1 前言

- shell編程屬於腳本編程，腳本文件就是指令的集合，GCC是GNU編譯系統驅動程序。
- Linux中的庫分兩種：靜態庫和共享庫。靜態庫以.a結尾，也叫歸檔文件（archive），類似於windows中的.lib文件，他的缺點是同時運行的多個程序使用同個函數庫函數時，內存中會有多個該函數及該程序文件的副本，浪費了內存。共享庫以.so結尾，類似於windows中的.Dll文件。
- shell是對linux內核的一種封裝，提供了一些函數的接口，shell程序設計就是利用這些函數接口進行編程。
- 一般linux系統的shell裝的是bash（Bourne Again Shell），安裝為/bin/sh，可以用\$/bin/bash --version命令查看bash版本號：

```
[root@localhost BashWorkStation]# /bin/bash --version
GNU bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

1.2.2 重定向

- 重定向有重定向輸入和重定向輸出，似乎重定向輸出用的比較多，可以將默認輸出到屏幕上的內容重定向輸出到一個文檔中，eg：

```
[ root@localhost BashWorkStation]# ls -al > lsoutput
[ root@localhost BashWorkStation]# more lsoutput
总用量 52
drwxr-xr-x.  2 root root 4096 8月  16 11:20 .
drwxr-xr-x. 15 root root 4096 8月  15 19:25 ..
-rw-r--r--.  1 root root   52 8月  15 19:25 bash1
-rwxr--r--.  1 root root   76 8月  15 19:25 bash2
-rwxr--r--.  1 root root  240 8月  16 10:22 CaseStructure
-rw-r--r--.  1 root root    0 8月  16 10:21 Casestructure
-rwxr-xr-x.  1 root root  357 8月  16 10:44 ChildFunction
-rwxr--r--.  1 root root   99 8月  15 19:39 findWord
-rwxr--r--.  1 root root  161 8月  16 09:42 ForStructure
-rwxr--r--.  1 root root  226 8月  16 09:23 GoodMorning
-rw-r--r--.  1 root root    0 8月  16 11:20 lsoutput
-rwxr--r--.  1 root root  168 8月  15 19:59 specialWord
-rwxr--r--.  1 root root  168 8月  16 09:10 TestCommand
-rwxr--r--.  1 root root  114 8月  16 09:54 UntilStructure
-rwxr--r--.  1 root root  181 8月  16 09:50 WhileStructure
```

- 也可以用>>將內容追加的重定向到已有的文件中，eg：

```
root@localhost BashWorkStation]# ps >>lsoutput
root@localhost BashWorkStation]# more lsoutput
总用量 52
drwxr-xr-x.  2 root root 4096 8月  16 11:20 .
drwxr-xr-x. 15 root root 4096 8月  15 19:25 ..
-rw-r--r--.  1 root root   52 8月  15 19:25 bash1
-rwxr--r--.  1 root root   76 8月  15 19:25 bash2
-rwxr--r--.  1 root root  240 8月  16 10:22 CaseStructure
-rw-r--r--.  1 root root    0 8月  16 10:21 Casestructure
-rwxr-xr-x.  1 root root  357 8月  16 10:44 ChildFunctionom
-rwxr--r--.  1 root root   99 8月  15 19:39 findWord
-rwxr--r--.  1 root root  161 8月  16 09:42 ForStructure
-rwxr--r--.  1 root root  226 8月  16 09:23 GoodMorning
-rw-r--r--.  1 root root    0 8月  16 11:20 lsoutput
-rwxr--r--.  1 root root  168 8月  15 19:59 specialWord
-rwxr--r--.  1 root root  168 8月  16 09:10 TestCommand
-rwxr--r--.  1 root root  114 8月  16 09:54 UntilStructure
-rwxr--r--.  1 root root  181 8月  16 09:50 WhileStructure
  PID TTY          TIME CMD
 69470 pts/0        00:00:00 sudo
 69478 pts/0        00:00:00 su
 69481 pts/0        00:00:00 bash
 72334 pts/0        00:00:00 ps
```

重定向輸入使用<符號

1.2.3 管道

- 管道的作用是可以讓進程進行通信，這樣一個進程的結果就可以作為另一個進程的操作對象，比如說想對ps列出的進程列表進行sort排序，然后分屏輸出，就可以使用命令：`ps | sort | more`

```
[root@localhost BashWorkStation]# ps | sort | more
69470 pts/0    00:00:00 sudo
69478 pts/0    00:00:00 su
69481 pts/0    00:00:00 bash
72339 pts/0    00:00:00 ps
72340 pts/0    00:00:00 sort
72341 pts/0    00:00:00 more
  PID TTY          TIME CMD
  _
```

■ 通配符

通配符可以匹配字符串、單個字符等

*：表示任意字符串

?：表示任意單個字符

[]：中可以添加任意的字符

[^]：表示不匹配[]裡面的字符

{ }：中添加的是字符串

1.2.4 創建腳本

- 可以使用vim編輯腳本文件，使用bash的話腳本文件的首行是：`#!/bin/sh` 聲明語句。
- linux中的變量不用聲明類型，系統默認是字符串型，當時數值時，系統會自動轉變類型，使用“\$”+變量名，可以訪問變量內容。要想輸出空格，要用“ ”括起來，否則空格會被會略。
- 可以使用read操作將命令行下用戶輸入的內容賦值給指定的變量。
- “” “ \ 對變量操作的區別：
- “\$變量名” 輸出的是變量的內容，
- ‘\$變量名’ \ \$變量名輸出的是變量名本身，也就是說他們兩個去掉了\$的作用。

```
[ root@localhost BashWorkStation]# cat specialWord
#!/bin/sh
myvar="Hello I am Jackson"
echo $myvar
echo "$myvar"
echo '$myvar'
echo \ $myvar

echo Enter a new string:
read myvar

echo '$myvar' now equals $myvar
exit 0
[ root@localhost BashWorkStation]# ./specialWord
Hello I am Jackson
Hello I am Jackson
$myvar
$myvar
Enter a new string:
I am Jackson Jackson
$myvar now equals I am Jackson Jackson
```

- 注意創建完腳本文件后需要把該文件的執行權限加上，命令為：chmod +x 文件名
- 在環境變量中，IFS表示輸入域分隔符，用戶可以將空格、制表符、換行符賦值給他，然后當shell讀取輸入時就可以安裝IFS的值分隔單詞，eg：

```
[ root@localhost BashWorkStation] # IFS=' '
[ root@localhost BashWorkStation] # set one two three
[ root@localhost BashWorkStation] # echo "$@"
one two three
[ root@localhost BashWorkStation] # echo "$*"
onetwothree
[ root@localhost BashWorkStation] # unset IFS
[ root@localhost BashWorkStation] # echo "$*"
one two three
[ root@localhost BashWorkStation] # █
```

- 可以看出 "\$@"是不受IFS影響的，"\$*"則受IFS影響。
- 布爾命令：**test**或者[]
- 條件語句后面跟的條件可以放在test或者[]中
- 模板為：
if test 條件
且等號左右都要有空格
if [條件] 要注意[]和條件語句之間要有空格隔開,並
- 然後
- 執行語句 執行語句
- fi fi
- 比較的條件可以有字符串、算術、文件相關測試，一下三個表都來自於《linux程序設計（第4版）》

字符串比较

结 果

string1 = string2

如果两个字符串相同则结果为真

string1 != string2

如果两个字符串不同则结果为真

-n string

如果字符串不为空则结果为真

-z string

如果字符串为null（一个空串）则结果为真

算术比较

结 果

expression1 -eq expression2

如果两个表达式相等则结果为真

expression1 -ne expression2

如果两个表达式不等则结果为真

expression1 -gt expression2

如果expression1大于expression2则结果为真

expression1 -ge expression2

如果expression1大于等于expression2则结果为真

expression1 -lt expression2

如果expression1小于expression2则结果为真

expression1 -le expression2

如果expression1小于等于expression2则结果为真

! expression

如果表达式为假则结果为真，反之亦然

文件条件测试	结 果
-d file	如果文件是一个目录则结果为真
-e file	如果文件存在则结果为真。要注意的是，历史上-e选项不可移植，所以通常使用的是-f选项
-f file	如果文件是一个普通文件则结果为真
-g file	如果文件的set-group-id位被设置则结果为真
-r file	如果文件可读则结果为真
-s file	如果文件的大小不为0则结果为真
-u file	如果文件的set-user-id位被设置则结果为真
-w file	如果文件可写则结果为真
-x file	如果文件可执行则结果为真

1.2.5 if語句

- 每組if語句都要使用fi做結束標志，eg：
- if 條件
- 然後
- 語句1
- else
- 語句2
-
- 其他
-
-

```
if 條件1
    語句1
elif 條件2
then
    語句2
    語句3
是
```



```
[ root@localhost BashWorkStation]# cat GoodMorning
#!/bin/sh
echo "is that moring? return yes or no: "
read answer
if [ "$answer" = "yes" ]
then
    echo "Good Morning !"
elif [ "$answer" = "no" ]
then
    echo "Good Afternoon !"
else
    echo "Sorry! Your input is error!"
    exit 1
fi
```

```
[ root@localhost BashWorkStation]# ./GoodMorning
is that moring? return yes or no:
yes
Good Morning !
[ root@localhost BashWorkStation]# ./GoodMorning
is that moring? return yes or no:
no
Good Afternoon !
```

1.2.6 for 語句

- for 語句一般針對於字符串
- 模板：for 變量名 in 值的範圍（一般為字符串）
- 做
- 語句
- 做完了

```
[root@localhost BashWorkStation]# cat ForStructure
#!/bin/sh
for foo1 in one two three
do
    echo "$foo1"
done

for foo2 in "one two three"
do
    echo "$foo2"
done
exit 0

[root@localhost BashWorkStation]# ./ForStructure
one
two
three
one two three
```

1.2.7 while語句和until語句

- while和until語句的區別是:
- while語句至少執行一次，而until語句則不一定；while是條件為真時執行循環體，until是當條件不成立時執行循環體（如果判斷某個用戶登錄的話用until語句比較合理）
- 模板：while (until) 條件
做
語句
做完了

```
[ root@localhost BashWorkStation]# cat WhileStructure
#!/bin/sh
echo "Please input your password: "
read answer
while [ "$answer" != "jiangjiang" ]
do
    echo "Sorry! Your input is incorrect, Please input again: "
    read answer
done
exit 0
[ root@localhost BashWorkStation]# ./WhileStructure
Please input your password:
jiang
Sorry! Your input is incorrect, Please input again:
jiang
Sorry! Your input is incorrect, Please input again:
jiangjiang
```

1.2.8 case語句

- case語句的每個條件成立后執行語句體結束時一定要記得用兩個;;表示結束，用esac表示case語句的結束
- 模板：case 變量名 in
- 值1) 語句體1
- 語句體2;;
- 值2) 語句體3;;
- —
- 埃薩克
- 出口0

```
[ root@localhost BashWorkStation]# cat CaseStructure
#!/bin/sh
echo "is that morning? yes or no "
read answer
case "$answer" in
    yes) echo "Good Morning!";;
    no) echo "Good Afternoon!";;
    y) echo "Good Morning!";;
    n) echo "Good Aternoon!";;
    *) echo "Your input is incorrect!";;
esac
exit 0
[ root@localhost BashWorkStation]# ./CaseStructure
is that morning? yes or no
y
Good Morning!
[ root@localhost BashWorkStation]# ./CaseStructure
is that morning? yes or no
n
Good Aternoon!
```

1.2.9 函數

- 函數不用聲明返回值類型，直接可以用：
- 函數名 () {
- }
- 來定義函數體。
- 如果函數需要操作從命令行輸入的變量時，在函數體中可以用 “\$*”來指代該參數。
- 其中調用函數時Child 后的\$1表示腳本程序的參數，判斷的是該函數的返回值
- 冒號:表示的是空命令，while true就等價於while

```
#!/bin/sh

Child(){
    echo "Is this your name? $*"
    while true
    do
        echo -n "Enter yes or no: "
        read answer
        case "$answer" in
            [yY]|[Yy][Ee][Ss]) return 0;;
            [nN]|[Nn][Oo]) return 1;;
            *) echo "Please answer yes or no!";;
        esac
    done
}

echo "Original parameters are $*"
if Child "$1"
then
    echo "Hi $1,nice name!"
else
    echo "Never mind!"
fi
exit 0
```

- 參考資料:<https://www.itdaan.com/tw/1a51cf5e08cf>
- 參考資料:<https://zh.wikipedia.org/wiki/Linux#%E6%AD%B7%E5%8F%B2>
- 參考資料:http://linux.vbird.org/linux_basic/0110whatislinux.php#whatislinux_unix