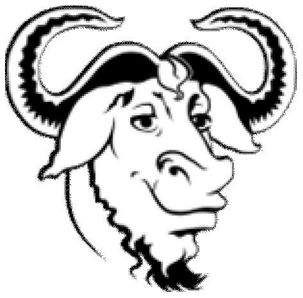
Linux操作系统基础教程



提到Linux操作系统，就不能不提UNIX和GNU。UNIX是由AT&T Bell实验室于1969年开发的多用户、多任务操作系统，是目前广泛使用的商业操作系统。而GNU则开始于1984年，Richard Stallman发起的GNU计划，目的是为了创建一套完全自由的操作系统（《GNU宣言》）。GNU是“GNU’s Not Unix”的递归缩写。（通用公共许可GNU General Public License，GPL，反版权copyleft, 自由软件基金会Free Software Foundation, FSF）。

1991年，Linus Torvalds编写了与UNIX兼容的LINUX操作系统并在GPL条款下发布。1992年，Linux与其他GNU软件结合，产生了完全自由的操作系统，所以Linux操作系统又被称为“GNU/Linux”。

三大主流shell

查看文件/etc/shell，可列出操作系统所包含的shell

Bourne shell——也称sh，是Version 7 Unix默认的Unix Shell。

Bourne Again Shell ——多数Linux操作系统的默认shell，是Bourne shell（UC Berleley）的扩展，是Bourne again/ Born again shell的双关语，1987年由Brian Fox创造。

C shell，TC shell——模仿C的语法，开发与BSD系统，脱胎于第六版UNIX的/bin/sh，也是Bourne shell的前身，加入了alias, command history等功能。

Korn shell是第一个UNIX shell，它完全向上兼容Bourne shell并包含了许多C shell的特性。

查看用户的默认shell，请查看文件/etc/passwd

每个进程都有一个进程号pid，可用pstree或者ps –ef命令查看

1. 常用Linux命令

（1） uname
（2） ls
（3） ps
（4） date

（5） **who**
（6） **pwd**
（7） **mkdir**
（8） **rmdir**
（9） **rm**
（10） **touch**
（11） **id**
（12） **group**
（13） **su**
（14） **which**
（15） **where**

2. C Shell

1. **shbang line**

#!/bin/csh 或者 #!/bin/tcsh

2. **comments**

```
# this is comment line
```

3. **wildcards**
4.
5. local variables

```
set variable=value
set tom=”tom”
```

6. global variables

```
setenv variable value
setenv tom tom
```

7. extract the variable value

```
echo $variable_name
echo $name
echo $PRINTER
```

8. user input

```
echo “what is your name?”
set name = $<
```

9. arguments

```
scriptname arg1 arg2 arg3
echo $1, $2, $3
echo $*
```

10. arrays

```
set word_list = ( word1, word2 word3 )
echo $world_list[2]
echo $word_list
echo $word_list[*]
```

11. command substitution
    a) Set variable_name = `command`

b) Echo $variable_name

12. arithmetic
    @ n = 5+5
    echo $n
13. operators
    Logic sign: ==, !=, >, >=, <, <=, &&, ||, !
14. conditional statements
    a) Loops
    b) File testing

–r	file readable by current user
-w	file writable by current user
-x	file executable by current user
-e	file exists
-o	file owned by current user
-z	file is zero length?
-d	file is directory?
-f	file is a plain file

3. Bourne shell

1. **the shbang line**
   #!/bin/sh
2. **comments**
   # this text is not
   # interpreted by the shell
3. **wildcards**
   *,?,
4. **displaying output**
   echo “What is your name?”
5. **local variables**
   variable_name=value
   name=”Maoying Woo”
   x=5
6. **Global variables**
   VARIABLE_NAME=value
   export VARIABLE_NAME
7. **extracting**
   echo $variable_name
8. **reading user input**
   echo “what is your name?”
   a) read name
   b) read name1 name2 name3
9. **arguments**
   a) scriptname arg1 arg2 arg3
   b) echo $1 $2 $3

10. arrays

```
set word1 word2 word3
echo $1 $2 $3
```

11. command substitution

``

12. arithmetic

```
n=`expr 5+5`
echo $n
```

13. operators

字符串运算： =, != string

数值运算： -e, -n

逻辑运算： -a,-o, !

关系运算： -gt, -ge, -lt, -le

14. conditional statements

similar to csh

4. Korn Shell (K Shell)

1. shbang line
   `#!/bin/ksh`
2. comments
   `# this program will test some files`
3. wildcards
   `rm *; ls ??; cat file[1-3];`
   `echo “How are you?”`
4. displaying output
   `echo “who are you?”`
   `print “how are you?”`
5. local variables
   `variable_name=value`
   `typeset variable_name=value`
   e.g., `name=”John Doe”`
6. global variable
   `export $VARIABLE_NAME=value`
   `export PATH=/bin:/usr/bin:.`
7. extracting values from variable name
   `echo $variable_name`
   `echo name`
8. reading user input
   `print –n “What is your name?”`
   `read name`
9. arguments
   `scriptname arg1 arg2 arg3`
   `echo $1 $2 $3`

10. arrays
    set apples pears peaches
    print $1 $2 $3
11. arithmetic
    typeset –i variable_name整型变量的设置
12. command substitution
    variable_name=`command`
    variable_name=$(command)
    echo $variable_name
13. operators
    string: =, !=
    number: ==, !=
    logic: &&, ||, !
    relation: >, >=, <, <=
14. conditional statements
15. loops
16. file testing

-d	file is a directory?
-a	file exists and is not a directory?
-r	file readable by current user?
-s	file of nonzero size?
-w	file writable by current user?
-x	file executable by current user?

5. Bourne Again shell

1. shbang line
   #!/bin/bash
2. comments
   # This is a comment line

3. wildcards
4. displaying output
5. local variables
   variable_name=value
   declare variable_name=value
6. global variables
   export VARIABLE_NAME=value
7. extracting values from variable_name
   echo ${variable_name_with_lengthy}
8. reading user input
   echo “what is your name?”
   read name

read name1 name2 name3

9. arguments
   $scriptname arg1 arg2 arg3
   echo $1 $2 $3
   echo $*
   echo $#
10. arrays
    set apples pears peaches
    echo $1 $2 $3
11. command substitution
    variable_name=`command`
    variable_name=$(command)
    echo $variable_name
12. arithmetic
    declare –i variable_name # integer variable
    typeset –i variable_name
    ((n=5+5))
    echo $n
13. operators
    等号: ==, !=
    logic: &&, ||, !
    relation: >, >=, <, <=
14. conditional statements
15. loops
16. functions

```
function_name() {
    block of code
}
function function_name {
    block of code
}
```

6. regular expression and pattern matching 正则表达式与模式匹配

1. regular expression

definition: a regular expression is just a pattern of characters used to match the same characters in a search.

```
Hi Tom,
I think I failed my anatomy test yesterday. I had a terrible stomachache. I
ate too many fried green tomatoes. Anyway, Tom, I need your help. I'd
like to make the test up tomorrow, but don't know where to begin
studying. Do you think you could help me? After work, about 7 PM, come
to my place and I'll treat you to pizza in return for your help. Thanks.
Your Pal,
guy@phantom
```

if we want to substitute “Tom” or “tom” in the text to “admin”, how?
vi里面寻找包含”tom”的行：/tom/，如果只是简单的寻找，一方面无法找到Tom，另一方面还会找到stomachache、tommorrow以及tomatoes等。因此需要比较复杂的正则表达式的应用。
:1,$s/\<[Tt]om\>/admin/g即可完成替代

- 其中1,$表示从第一行到最后一行
- s表示取代（substitute）
- [Tt]表示字母”T”或者”t”的其中一个
- \<表示一个单词的起始，而\>表示一个单词的结束，这里不一定是同一个单词
- /g表示全局取代

2. metacharacter元字符

^	Beginning of line anchor	/^love/	Matches all lines beginning with love
$	End of line anchor	/love$/	Matches all lines ending with love
.	Matches one character	/l..e/	Matches lines containing an l, Followed by two characters
*	Matches zero or more of the preceding	/ *love/	Matches lines with zero or more spaces, followed by the pattern love
[ ]	Matches one in the set	/[Ll]ove/	Matches line containing love or Love
[x-y]	Matches one character within range from x to y	/[A-Z]ove/	Matches letters from A through Z followed by ove
[^]	Matches one character not in the set	/[^A-Z]/	Matches any character not in the range between A and Z
\	Used to escape a meacharacter	/love\./	Matches lines containing love, followed by a literal period; normally the period matches a single any character
\<	Beginning of word anchor	/\<love/	Matches lines containing a word that begins with love (supported by vi and grep)
\>	End-of-word anchor	/love\>/	Matches lines containing a word that

			ends with love (supported by vi and grep)
\(..\)	Tags match characters to be used	/\(love\)able \1er/	May use up to nine tags, starting with the first tag at the leftmost part of the pattern
x\{m\} or x\{m,\} or x\{m,n\}		o\{5,10\}	Matches if line contains between 5 to 10 consecutive occurrence of the letter o (supported by vi and grep)

example:

I had a lovely time on our little picnic. Lovers were all around us. It is springtime. Oh love, how much I adore you. Do you know the extent of my love? Oh, by the way, I think I lost my gloves somewhere out in that field of clover. Did you see them? I can only hope love is forever. I live for you. It's hard to get back in the groove.

/^love/
/love$/
/l.ve/ 中，dot(.)匹配除了换行符之外的任意字符，如live/love/leve/lave
/o*ve/ 中，asterisk(*)匹配0个到多个前一个字符o，包括ve/ove/oove/…
/[Ll]ove/匹配love/Love
/ove[a-z]/匹配ovea/oveb/…/ovez
/ove[^A-Za-z0-9]/匹配与ove紧跟的字符不是大写、小写字母和数字，例如"ove "(空格)

3. 更为复杂的元字符

```
Christian Scott lives here and will put on a Christmas party. There are
around 30 to 35 people invited.
They are:
                                        Tom
Dan
Rhonda Savage
Nicky and Kimberly
Steve, Suzanne, Ginger and Larry.
```

/^[A-Z]..$/匹配大写字母开始，长度为3个字符的行
/^[A-Z][a-z]*3[0-5]/大写字母开始，紧跟0个或多个小写字母或空格，再紧跟数字3和0-5之间数字的行
/^ *[A-Z][a-z][a-z]$/匹配0个到多个空格开始，后面紧跟一个大写字母和两个小写字母的行
/[a-z]*\./匹配包含0-多个小写字母，后面紧跟句号的行
/^[A-Za-z]*[^,][A-Za-z]*$/呢？

4. more regular expression

```
$ cat textfile
Unusual occurrences happened at the fair.
Patty won fourth place in the 50 yard dash square and fair.
Occurences like this are rare.
The winning ticket is 55222.
The ticket I got is 54333 and Dee got 55544.
Guy fell down while running around the south bend in his last event.
```

- 在每一行查找单词fourth
- 查找以f开始的单词，后面紧跟0-多个字符，最后是”th”的字符串的行
- 将所有occurence 和 Occurence 修改为occurrence和Occurrence

  :1,$s/\([Oo]ccur/)ence/\1rence/g
- 颠换短句”fair and square”中的单词fair 和 square

  :1,$s/\(fair\) and \(square\)/\2 and \1/g
- 查找包含连续两个5，紧跟3个2，后面是句号

  :1,$/5\{2\}2\{3\}./

5. grep family继续巩固正则表达式：来源于ex编辑器

grep是常用的查找pattern的命令

ex编辑器

:/**RE**/p　　　打印包含pattern的第一行

:g/**RE**/p　　　　　打印所有包含pattern的行

工作模式

在一个或多个文件中查找多个字符模式，如果存在空格，模式必须用引号

用法

grep word filename1 filename2

如在/etc/passwd中查找admin这个用户： grep admin /etc/passwd

常用于管道，例如ps ef | grep bash

grep选项列表

Options	What it does
-b	找到匹配的位置在行前输入block number
-c	输出匹配行的数目， 而不输出结果
-h	不显示文件名
-i	不考虑匹配字母的大小写
-l	列出具有匹配行的文件名称，用分行符分割
-n	在前面输出行号，紧跟该行内容
-s	Silently没有输出，除非有错误产生，用来检查退出状态，找到返回0，找不到模式退出状态为1，找不到文件报错退出状态为2；可用echo $?检查状态
-v	Inverts，仅仅输出不匹配的行
-w	查找表达式作为一个单词，把它视作\<和\>内部

6. **examples**
   - → grep NW datafile
   - → grep NW d*
   - → grep ‘^n’ datafile
   - → grep ‘4$’ datafile
   - → grep ‘TB Savage’ datafile
   - → grep ‘5\..’ datafile
   - → grep ‘\.5’ datafile
   - → grep ‘^[we]’ datafile
   - → grep ‘[^0-9]’ datafile
   - → grep ‘[A-Z][A-Z] [A-Z]’ datafile
   - → grep ‘ss* ’ datafile
   - → grep ‘[a-z]\{5,9\}’ datafile
   - → grep ‘\<north’ datafile
   - → grep ‘\<north\>’ datafile
   - → grep ‘[a-z].*n\>’ datafile
   - → grep –n ‘^south’ datafile
   - → grep –i ‘pat’ datafile
   - → grep –v ‘Suan Chin’ datafile
   - → grep –v ‘Suan Chin’ datafile > tempfile
   - → mv temp datafile
   - → grep –l ‘SE’ *
   - → grep –c ‘west’ datafile
   - → grep –w ‘north’ datafile
   - → ls –l | grep ‘^d’
   - → rpm –qa | grep http

metacharacter

+	matches one or more of the preceding character	‘[a-z]+ove’
?	matches zero or one of the preceding character	‘lo?ve’
a|b	matches either a or b	‘love|hate’
()	group characters	‘love(able|ly)’ ‘(ov)+’

egrep

egrep ‘^ +’ file	prints lines beginning with one or more spaces
egrep ‘^ *’ file	prints lines beginning with zero or more spaces
egrep ‘(Tom|Dan) Savage’	prints lines containing Tom Savage or Dan Savage
egrep ‘(ab)+’ file	prints lines with one or more occurrences of ab
egrep ‘^X[0-9]?’	prints lines beginning with X followed by 0-1 single digit

fgrep: fixed grep, does not recognize any regular expression metacharacters as being special.

grep –R recursively descend a directory tree

rgrep

linux GNU grep

grep ‘pattern’ filename 基本方式
grep –G ‘pattern’ filename 基本方式
grep –E ‘pattern’ filename 扩展RE元字符
grep –F ‘pattern’ filename 无RE元字符，同fgrep
grep –P ‘pattern’ filename 理解为Perl-RE

\w	表示[a-zA-Z0-9_]
\W	表示[^a-zA-Z0-9_]
\b	表示边界，例如\blove\b匹配单词love

- egrep ‘NW|EA’ datafile
- grep –E ‘NW|EA’ datafile
- grep ‘NW|EA’ datafile×
- grep ‘NW\|EA’ datafile

- egrep ‘3+’ datafile
- grep –E ‘3+’ datafile
- grep ‘3\+’ datafile

- egrep ‘2\.?[0-9]’ datafile
- grep –E ‘2\.?[0-9]’ datafile
- grep ‘2\.\?[0-9]’ datafile

grep –V显示版本号
grep -2 显示pattern的前后两行
grep –C 2 前后两行
grep –A 2 后面两行
grep –B 2 后面两行