# LINUX 作業系統實務

# 09. Regular Expressions – grep and sed

2020 TKU

Sherry Yin

# 正規表達式 Regular Expression: regex、regexp或RE

- 使用單個字串來描述、匹配一系列符合某個句法規則的字串。
- 通常被用來檢索、替換那些符合某個模式的文字。
- 正規表達式這個概念最初是由Unix中的工具軟體（例如sed和grep）普遍開的。
- 1940年，沃倫·麥卡洛克與Walter Pitts將神經系統中的神經元描述成小而簡單的自動控制元。
- 1950年代，數學家史蒂芬·科爾·克萊尼利用稱之為「正規集合」的數學符號來描述此模型
- 肯·湯普遜將此符號系統引入編輯器QED，隨後是Unix上的編輯器ed，並最終引入grep。自此以後，正規表達式被廣泛地應用於各種Unix或類Unix系統的工具中。
- 正規表示式的POSIX規範，分為基本型正規表示式（Basic Regular Expression，BRE）和擴充型正規表示式（Extended Regular Express，ERE）兩大流派。

# 理論

- 正規表示式可以用形式化語言理論的方式來表達。正規表示式由常數和算子組成，它們分別表示字串的集合和在這些集合上的運算。給定有限字母表Σ定義了下列常數：

  - 空集∅表示集合∅。[發音: u in ‘burn’]
  - 空字串ε表示僅包含一個「不含任何字元、長度為0的字串」的集合。[發音:epsilon]
  - 文字字元$a \in \Sigma$表示僅包含一個元素a的集合{a}。[發音:sigma]

# 運算

- 串接 RS表示集合{αβ |α∈ R, β∈ $S$}}，這裡的αβ表示將α和β兩個字串按順序連接。例如：{ab,c}{d,ef}={abd,abef,cd,cef}。
- 選擇 R|S表示R和S的聯集。例如：{ab,c}|{ab,d,ef}={ab,c,d,ef}。
- 克萊尼(Kleene)星號 R*表示包含ε且在字串串接運算下閉合的R的最小超集。這是可以通過R中零或有限個字串的串接得到所有字串的集合。例如： {ab,c}*={ε,ab,c,abab,abc,cab,cc,abababab,...}
- 上述常數和算子形成了克萊尼代數。
- 也很常使用U, + 或V替代豎線。

# 優先級順序

- 為了避免括號，假定Kleene星號有最高優先級，接著是串接，接著是聯集。
- 如果沒有歧義則可以省略括號。例如：(ab)c可以寫為abc而a|(b(c*))可以寫為a|bc*。
- 舉例:
  - a|b*表示{ε,a,b,bb,bbb,...}
  - (a|b)*表示包括空字串和任意數目個a或b字元組成的所有字串的集合：{ε,a,b,aa,ab,ba,bb,aaa... }。
  - ab*(c|ε)表示開始於一個a接著零或多個b和最後一個可選的c組成的字串的集合：{a,ac,ab,abc,abb,abbc...}。
- 正規表示式也定義了?和+；aa*等於a+，表示a出現至少一次；而(a|ε)等於a?，表示a出現1次或不出現。
- ~為補運算子, ~R表示在Σ *上但不在R中的所有字串的集合。

# 基本語法

- 一個正規表示式通常被稱為一個模式（pattern），為用來描述或者匹配一系列符合某個句法規則的字串。例如：Handel、Händel和Haendel這三個字串，都可以由H(a|ä|ae)ndel這個模式來描述。
- 大部分正規表示式的形式都有如下的結構：
  - 選擇: 豎線|代表選擇（即或集），具有最低優先級。例如gray|grey可以匹配grey或gray。
  - 數量限定: 某個字元後的數量限定符用來限定前面這個字元允許出現的個數。最常見的數量限定符包括+、?和*（不加數量限定則代表出現一次且僅出現一次）：
    - 加號+代表前面的字元必須至少出現一次。（1次或多次）。例如，goo+gle可以匹配google、gooogle、goooogle等;
    - 問號?代表前面的字元最多只可以出現一次。（0次或1次）。例如，colou?r可以匹配color或者colour;
    - 星號*代表前面的字元可以不出現，也可以出現一次或者多次。（0次、1次或多次）。例如，0*42可以匹配42、042、0042、00042等。

# 匹配

- 圓括號()可以用來定義運算子的範圍和優先度。例如，gr(a|e)y等價於gray|grey，(grand)?father匹配father和grandfather。
- 上述這些構造子都可以自由組合，因此H(ae?|ä)ndel和H(a|ae|ä)ndel是相同的，表示{"Handel", "Haendel", "Händel"}。

# PCRE（Perl_Compatible_Regular_Expressions）表達式全集

- \ 將下一個字元標記為一個特殊字元等, 例如，「\n」匹配一個換行符。序列「\\」匹配「\」而「\(」則匹配「(」。
- ^ 匹配輸入字串的開始位置。如果設定了RegExp物件的Multiline屬性，^也匹配「\n」或「\r」之後的位置。
- $ 匹配輸入字串的結束位置。如果設定了RegExp物件的Multiline屬性，$也匹配「\n」或「\r」之前的位置。
- * 匹配前面的子表達式零次或多次。例如，zo*能匹配「z」、「zo」以及「zoo」。*等價於{0,}。
- + 匹配前面的子表達式一次或多次。例如，「zo+」能匹配「zo」以及「zoo」，但不能匹配「z」。+等價於{1,}。
- ? 匹配前面的子表達式零次或一次。例如，「do(es)?」可以匹配「does」中的「do」和「does」。?等價於{0,1}。
- {n} n是一個非負整數。匹配確定的n次。例如，「o{2}」不能匹配「Bob」中的「o」，但是能匹配「food」中的兩個o。
- {n,} n是一個非負整數。至少匹配n次。例如，「o{2,}」不能匹配「Bob」中的「o」，但能匹配「foooood」中的所有o。「o{1,}」等價於「o+」。「o{0,}」則等價於「o*」。
- {n,m} m和n均為非負整數，其中n<=m。最少匹配n次且最多匹配m次。例如，「o{1,3}」將匹配「fooooood」中的前三個o
- ? 非貪心量化（Non-greedy quantifiers）：當該字元緊跟在任何一個其他重複修飾詞（*,+,?，{n}，{n,}，{n,m}）後面時，匹配模式是非貪婪的。非貪婪模式儘可能少的匹配所搜尋的字串，而預設的貪婪模式則儘可能多的匹配所搜尋的字串。例如，對於字串「oooo」，「o+?」將匹配單個「o」，而「o+」將匹配所有「o」。

- . 匹配除「\r」「\n」之外的任何單個字元。要匹配包括「\r」「\n」在內的任何字元，請使用像「(.|\r|\n)」的模式。
- (pattern) 匹配pattern並取得這一匹配的子字串。
- (?:pattern) 匹配pattern但不取得匹配的子字串（shy groups），也就是說這是一個非取得匹配，不儲存匹配的子字串用於向後參照。這在使用或字元「(|)」來組合一個模式的各個部分是很有用。例如「industr(?:y|ies)」就是一個比「industry|industries」更簡略的表達式。
- (?=pattern) 正向肯定預查（look ahead positive assert），在任何匹配pattern的字串開始處匹配尋找字串。這是一個非取得匹配，也就是說，該匹配不需要取得供以後使用。例如，「Windows(?=95|98|NT|2000)」能匹配「Windows2000」中的「Windows」，但不能匹配「Windows3.1」中的「Windows」。預查不消耗字元，也就是說，在一個匹配發生後，在最後一次匹配之後立即開始下一次匹配的搜尋，而不是從包含預查的字元之後開始。
- (?!pattern) 正向否定預查（negative assert），在任何不匹配pattern的字串開始處匹配尋找字串。這是一個非取得匹配，也就是說，該匹配不需要取得供以後使用。例如「Windows(?!95|98|NT|2000)」能匹配「Windows3.1」中的「Windows」，但不能匹配「Windows2000」中的「Windows」。預查不消耗字元，也就是說，在一個匹配發生後，在最後一次匹配之後立即開始下一次匹配的搜尋，而不是從包含預查的字元之後開始
- (?<=pattern) 反向（look behind）肯定預查，與正向肯定預查類似，只是方向相反。例如，「(?<=95|98|NT|2000)Windows」能匹配「2000Windows」中的「Windows」，但不能匹配「3.1Windows」中的「Windows」。
- (?<!pattern) 反向否定預查，與正向否定預查類似，只是方向相反。例如「(?<!95|98|NT|2000)Windows」能匹配「3.1Windows」中的「Windows」，但不能匹配「2000Windows」中的「Windows」。

- x|y　　沒有包圍在()里，其範圍是整個正規表示式。例如，「z|food」能匹配「z」或「food」。「(?:z|f)ood」則匹配「zood」或「food」。
- [xyz]　字元集合（character class）。匹配所包含的任意一個字元。例如，「[abc]」可以匹配「plain」中的「a」。特殊字元僅有反斜線\保持特殊含義，用於跳脫字元。其它特殊字元如星號、加號、各種括號等均作為普通字元。脫字元^如果出現在首位則表示負值字元集合；如果出現在字串中間就僅作為普通字元。連字元 - 如果出現在字串中間表示字元範圍描述；如果如果出現在首位（或末尾）則僅作為普通字元。右方括號應跳脫出現，也可以作為首位字元出現。
- [^xyz]　排除型字元集合（negated character classes）。匹配未列出的任意字元。例如，「[^abc]」可以匹配「plain」中的「plin」。
- [a-z]　字元範圍。匹配指定範圍內的任意字元。例如，「[a-z]」可以匹配「a」到「z」範圍內的任意小寫字母字元。
- [^a-z]　排除型的字元範圍。匹配任何不在指定範圍內的任意字元。例如，「[^a-z]」可以匹配任何不在「a」到「z」範圍內的任意字元。
- [:name:]　　增加命名字元類（named character class）[註 1]中的字元到表達式。只能用於方括號表達式。
- [=elt=]　增加目前locale下排序（collate）等價於字元「elt」的元素。例如，[=a=]可能會增加ä、á、à、ă、ắ、ằ、ẵ、ẳ、â、ấ、ầ、ẫ、ẩ、ǎ、å、ǻ、ǟ、ã、ȧ、ā̄、ą、ā、ả、ȁ、â、ạ、ặ、ậ、ᶏ、ⱥ、ᶐ、ɐ、ɑ。只能用於方括號表達式。
- [.elt.]　增加排序元素（collation element）elt到表達式中。這是因為某些排序元素由多個字元組成。例如，29個字母表的西班牙語， "CH"作為單個字母排在字母C之後，因此會產生如此排序「cinco, credo, chispa」。只能用於方括號表達式。
- \b　　匹配一個單詞邊界，也就是指單詞和空格間的位置。例如，「er\b」可以匹配「never」中的「er」，但不能匹配「verb」中的「er」。
- \B　　匹配非單詞邊界。「er\B」能匹配「verb」中的「er」，但不能匹配「never」中的「er」。

- \cx　匹配由x指明的控制字元。x的值必須為A-Z或a-z之一。否則，將c視為一個原義的「c」字元。控制字元的值等於x的值最低5位元（即對3210進位的餘數）。例如，\cM匹配一個Control-M或回車字元。\ca等效於\u0001, \cb等效於\u0002, 等等…
- \d　匹配一個數字字元。等價於[0-9]。注意Unicode正規表示式會匹配全形數字字元。
- \D　匹配一個非數字字元。等價於[^0-9]。
- \f　匹配一個換頁符。等價於\x0c和\cL。
- \n　匹配一個換行符。等價於\x0a和\cJ。
- \r　匹配一個回車字元。等價於\x0d和\cM。
- \s　匹配任何空白字元，包括空格、制表符、換頁符等等。等價於[ \f\n\r\t\v]。注意Unicode正規表示式會匹配全形空格符。
- \S　匹配任何非空白字元。等價於[^ \f\n\r\t\v]。
- \t　匹配一個制表符。等價於\x09和\cI。
- \v　匹配一個垂直制表符。等價於\x0b和\cK。
- \w　匹配包括底線的任何單詞字元。等價於「[A-Za-z0-9_]」。注意Unicode正規表示式會匹配中文字元。

- \W　匹配任何非單詞字元。等價於「[^A-Za-z0-9_]」。
- \xnn　十六進位跳脫字元序列。匹配兩個十六進位數字nn表示的字元。例如，「\x41」匹配「A」。「\x041」則等價於「\x04&1」。正規表達式中可以使用ASCII編碼。.
- \num　向後參照（back-reference）一個子字串（substring），該子字串與正規表示式的第num個用括號圍起來的捕捉群（capture group）子表達式（subexpression）匹配。其中num是從1開始的十進位正整數，其上限可能是9[註 2]、31[註 3]、99甚至無限[註 4]。例如：「(.)\1」匹配兩個連續的相同字元。
- \n　標識一個八進位跳脫值或一個向後參照。如果\n之前至少n個取得的子表達式，則n為向後參照。否則，如果n為八進位數字（0-7），則n為一個八進位跳脫值。
- \nm　3位八進位數字，標識一個八進位跳脫值或一個向後參照。如果\nm之前至少有nm個獲得子表達式，則nm為向後參照。如果\nm之前至少有n個取得，則n為一個後跟文字m的向後參照。如果前面的條件都不滿足，若n和m均為八進位數字（0-7），則\nm將匹配八進位跳脫值nm。
- \nml　如果n為八進位數字（0-3），且m和l均為八進位數字（0-7），則匹配八進位跳脫值nml。
- \un　Unicode跳脫字元序列。其中n是一個用四個十六進位數字表示的Unicode字元。例如，\u00A9匹配著作權符號（©）。

# Unicode處理與字元組

- 在.NET、Java、JavaScript、Python的正規表示式中，可以用\uXXXX表示一個Unicode字元，其中XXXX為四位16進位數字。
- 更詳細請參照: https://zh.m.wikipedia.org/zh-tw/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F
- http://alfredwebdesign.blogspot.com/2014/08/php-regular-expression.html
- http://www.regular-expressions.info/php.html

# 玩遊戲

- Fill a grid with characters that match the regexes on the vertical and horizontal lines, usually revealing a word or phrase https://regexcrossword.com/
- Find a regex that matches all highlighted elements from a piece of content http://play.inginf.units.it/
- Find a regex that matches a list of words, but it also must not match another list https://alf.nu/RegexGolf

# Simple filters

- Filters: text manipulation tools, commands that use both standard IO.
- $ pr –t –n –d –o 10 group1

  1 root:x:0:root
  2 bin:x:1:root,bin,daemon
  3 users:x:200:henry,image,enquiry
  4 adm:x:25:adm,daemon,listen

  ....(pr is often used with | lp to preprocess text files before they are printed.
- -t: Eliminates headers, footers, and margins totally
- -n: Numbers lines in output
- -d: Double-spaces output
- -o n: Offsets output by n spaces

# cmp: Byte-by-Byte Comparison

- $ cmp group1 group1
  group1 group2 differ: char 47, line 3

  It echoes the first mismatch only by default.
- $ cmp -l group[12]  Using a wild card

  | | | |
  |---|---|---|
  | 47 | 62 | 61 |
  | 109 | 70 | 71 |
  | 128 | 71 | 70 |

  cmp: EOF on group1  group1 finishes first

  Three differences. Character 47 has the ASCII octal values 62 and 61 in the two files.
- $ cmp –l group? | wc –l  Count the number of differences
  3  3 differences till EOF
- This command returns true when files are identical and false otherwise.

# comm: What is Common?

- Compares files line by line and displays results in three columns:
  - Column 1 Lines unique to the first file
  - Column 2 Lines unique to the second file
  - Column 3 Lines common to both files
- Files must be sorted first.
- $ comm -3 foo1 foo2 Selects lines not common to both files
- $ comm -13 foo1 foo2 Selects lines present only in second file

# diff: Converting One File to Another

- $ diff group[12]

```
3c3                                          Change line3 of first file
< users:x:200:henry,image,enquiry      Change this line
---                                                  to
> users:x:100:henry,image,enquiry      this
5,6c5,6                                      Change lines 5 to 6
< dialout:x:18:root,henry                    Replace these two lines
< lp:x:19:1p
---
< dialout:x:18:root,henry                    with these two
< lp:x:19:1p
7a8                                          Append after line 7 of first file
> cron:x:16:cron                             this line
```

# diff instructions


- 3c3: changes line 3 with one line, which remains line 3 afterwards.
- 7a8: appends a line after line 7, yielding line number 8 in the second change.
- 5,6c: changes two lines
- $ diff –e

  This command option produces a set of instructions only

# head: Displaying the Beginning of a File

- It displays 10 lines by default.
- $ head -n 3 group1

  displays the first 3 lines
- $ vi `ls -t | head -n 1`

  picks up the first file from the list of files displayed in order of their modification time.
- $ grep “IMG SRC.*GIF” quote.html | head –n 5

  picks up the first five lines containing the string GIF after the words IMG SRC

# tail: Displaying the End of a File

- It displays the last 10 lines by default.
- $ tail -n 3 group1
  displays the last 3 lines
- $ tail +801 foo
  801th line onwards, possible with + symbol
- $ tail –f /oracle/app/oracle/product/8.1/orainst/install.log
  monitor the installation of Oracle by watching the growth of the log file.
- $ tail -c -512 foo  Copies last 512 bytes from foo
- $ tail -c +512 foo  Copies everything after 512 bytes from foo

# cut: Slitting a File Vertically

- head and tail are used to slice a file horizontally, you can slice a file vertically with the cut command.
- $ cut -c1-4 group1 extract the first four columns of the group file (1-4 is the range)
- $ cut -c -3,6-22,28-34,55- foo (-3 is the same as 1-3, 55- means from 55 to the end of the line, notice that this must be an ascending list)

# sort: Ordering a File

- Ordering of data in ascending or descending sequence.
- -tchar: uses delimiter char to identify fields
- -k n: sorts on nth field
- -k m,n: starts sort on mth field and ends sort on nth field
- -u: removes repeated lines
- -n: sorts numerically
- -r: reverse sort order

# uniq: Locate Repeated and Non-repeated Lines

- $ cat dept.lst (this is a file containing repeated lines)
  01: 123
  02: 2123
  03: 3123
  03: 3123
  04: 4123
- $ uniq dept.lst (uniq fetches a copy of each line and writes it to the SO)
  01: 123
  02: 2123
  03: 3123
  04: 4123
- $ sort dept.lst | uniq - uniqlist (same output, writes to uniqlist)
- -u: selecting the non-repeated lines
- -d: selecting the duplicate lines
- -c: counting frequency of occurrence
  $ cut -d: -f3 shortlist | sort | uniq -c (cut the third field with cut, sort it with sort, and then run uniq -c to produce a count)

# tr: Translating Characters

- format: tr options expression1 expression2 standard input
- Let's use tr to replace the : with a ~ and the / with a -.
  $ tr ':/' '~-' < shortlist | head -n 3  (the length of the two expressions should be equal)
- Also can change case of text:
  $ head -n 3 shortlist | tr '[a-Z]' '[A-Z]'
- Delete characters (-d):
  $ tr -d ':/' < shortlist | head -n 3
- Compress multiple consecutive characters (-s):
  $ tr -s ' ' < shortlist | head -n 3
- Complement values of expression (-c):
  $ tr -cd ':/' < shortlist  (delete all characters except the : and /)

# Example I: listing the give large files in the current directory

1. Reverse-sort this space-delimited output in numeric sequence on the fifth field:

   $ ls -l | sort -k 5 -nr

2. Extract the first five lines from the sorted output:

   $ ls -l | sort -k 5 -nr | head -n 5

3. Squeeze multiple spaces to a single space:

   $ ls -l | sort -k 5 -nr | head -n 5 | tr -s " "

4. Cut the fifth and last fields:

   $ ls -l | sort -k 5 -nr | head -n 5 | tr -s " " | cut -d" " -f5,9

# Example II: creating a word-usage list

1. tr can place each word in a separate line:
   $ tr " \011" "\012\012" < foo1  (space is \040, tab is octal 011)
2. Delete all non-alphabetic characters (apart from the newline), need to use complementary (-c) and delete (-d):
   $ tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]"
3. Sort this output (each word on a separate line), pipe it to uniq -c:
   $ tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]" | sort | uniq -c
4. Sort the list in reverse numeric sequence and print it in 3 cols:
   $ tr " \011" "\012\012" < foo1 | tr -cd "[a-zA-Z\012]" | sort | uniq -c \
   > sort -nr | pr -t -3
   (split the command line into two lines by using \)

# Example III: finding out the difference between two password files

1. First cut out the first field of passwd1 and save the sorted output:

   $ cut -f1 -d: passwd1 | sort > temp

   $ cut -f1 -d: passwd2 | sort > temp2

2. Compare these two files with comm -23 (can do this job without creating the temp2):

   $ cut -f1 -d: passwd2 | sort | comm -23 temp - ; rm temp

   ftp

   joe

   juliet

   (comm -23 lists only those lines that are in the first file, and the - ensured that the output from sort was supplied as SI)

# grep: Searching for a Pattern

- Format: grep options pattern filename(s)
- Example: use grep to display lines containing the string sales from the sample emp.lst

```
$ cat emp.lst
2233|charles harris  |g.m.     |sales     |12/12/52| 90000
9876|bill johnson    |director |production|03/12/50|130000
5678|robert dylan    |d.g.m.   |marketing |04/19/43| 85000
2365|john woodcock   |director |personnel |05/11/47|120000
5423|barry wood      |chairman |admin     |08/30/56|160000
1006|gordon lightfoot|director |sales     |09/03/38|140000
6213|michael lennon  |g.m.     |accounts  |06/05/62|105000
1265|p.j. woodhouse  |manager  |sales     |09/12/63| 90000
4290|neil o'bryan    |executive|production|09/07/50| 65000
2476|jackie wodehouse|manager  |sales     |05/01/59|110000
6521|derryk o'brien  |director |marketing |09/26/45|125000
3212|bill wilcocks   |d.g.m.   |accounts  |12/12/55| 85000
3564|ronie trueman   |executive|personnel |07/06/47| 75000
```

```
$ grep sales emp.lst
2233|charles harris  |g.m.     |sales     |12/12/52| 90000
1006|gordon lightfoot|director |sales     |09/03/38|140000
1265|p.j. woodhouse  |manager  |sales     |09/12/63| 90000
2476|jackie wodehouse|manager  |sales     |05/01/59|110000
```

- grep is also a filter, so it can search its SI for the pattern and store the output in a file:

  $ who | grep henry > foo

- grep can search two files:

```
$ grep director emp1.lst emp2.lst
emp1.lst:1006|gordon lightfoot|director |sales     |09/03/38|140000
emp1.lst:6521|derryk o'brien  |director |marketing |09/26/45|125000
emp2.lst:9876|bill johnson    |director |production|03/12/50|130000
emp2.lst:2365|john woodcock   |director |personnel |05/11/47|120000
```

(To suppress the filenames, you can use cut to select all but the first field using grep as its input. You can also use: cat emp[12].lst | grep “director”

# Quoting in grep

- Let's use a two-word string both within and without quotes:

```
$ grep gordon lightfoot emp.lst
grep: lightfoot: No such file or directory
emp.lst:1006|gordon lightfoot|director |sales      |09/03/38|140000
$ grep 'gordon lightfoot' emp.lst
1006|gordon lightfoot|director |sales      |09/03/38|140000
```

# grep Options

| Option | Significance |
|---|---|
| -c | Displays count of number of occurrences |
| -l | Displays list of filenames only |
| -n | Displays line numbers along with lines |
| -v | Doesn't display lines matching expression |
| -i | Ignores case when matching |
| -h | Omits filenames when handling multiple files |
| -w | Matches complete word (**grep** *only*) |
| -e *pat* | Also matches pattern *pat* beginning with a - (hyphen) |
| -e *pat* | As above, but can be used multiple times *(Linux and some UNIX versions)* |
| -E | Treats pattern as an **egrep** regular expression *(Linux and Solaris-xpg4)* |
| -F | Matches pattern in **fgrep**-style *(Linux and Solaris-xpg4)* |
| -*n* | Displays line and *n* lines above and below *(Linux only)* |
| -A *n* | Displays line and *n* lines after matching lines *(Linux only)* |
| -B *n* | Displays line and *n* lines before matching lines *(Linux only)* |
| -f *file* | Take patterns from *file*, one per line *(Linux only)* |

- -v: deleting lines
  $ grep -v 'director' emp.lst > otherlist
  $ wc -l otherlist
  11 otherlist
- -l: displaying filenames: to locate files where a variable or system call has been used:
  $ grep -l fork *.c
  fork.c: printf("before fork\n");
  orphan.c: if ((pid = fork()) > 0)
  ...
- -e: matching multiple patterns:
  $ grep -e woodhouse -e wood -e woodcock emp.lst
- -e: for patterns beginning with a -:
  $ grep -e "-mtime" /var/spool/cron/crontabs/*
- Locates all C programs in $HOME that contain the line "#include <fcntl.h>":
  $ find $HOME -name "*.c" -exec grep -l "#include <fcntl.h>" {} \; > foo
  $ find $HOME -name "*.c" -exec grep -l "#include <fcntl.h>" {} /dev/null \;
- -n: displays a certain number of lines above and below it.
  $ grep -1 "foreach" count.pl

# Basic Regular Expressions (BRE)

- POSIX identifies regular expressions as two categories: basic and extended.
- grep supports BRE by default, and ERE with the -E option.
- sed supports only the BRE.

# BRE Character Set used by grep, sed, and awk

| Pattern | Matches |
|---|---|
| * | Zero or more occurrences of previous character |
| g* | Nothing or g, gg, ggg, etc. |
| gg* | g, gg, ggg, etc. |
| . | A single character |
| .* | Nothing or any number of characters |
| [*pqr*] | A single character *p*, *q* or *r* |
| [abc] | a, b or c |
| [*c1-c2*] | A single character within the ASCII range represented by *c1* and *c2* |
| [1-3] | A digit between 1 and 3 |
| [^*pqr*] | A single character which is not a *p*, *q* or *r* |
| [^a-zA-Z] | A nonalphabetic character |
| ^*pat* | Pattern *pat* at beginning of line |
| *pat*$ | Pattern *pat* at end of line |

| | |
|---|---|
| g* | Nothing or g, gg, ggg, etc |
| gg* | g, gg, ggg, etc |
| .* | Nothing or any number of chars |
| [1-3] | A digit between 1 and 3 |
| [^a-zA-Z] | A nonalphabetic char |
| bash$ | bast at end of line |
| ^base$ | bash as the only word in line |
| ^$ | Lines containing nothing |

Note: the pattern [a-zA-Z0-9] matches a single alphanumeric character.

^ 插入符 is for negating a class.

# The *: immediately preceding character

- It indicates that the previous character can occur many times, or not at all.
- ee* matches a string beginning with e, not e*
- To match trueman and Truman:

  $ grep “true*man” emp.lst
- To match wilcocks and wilcox:

  $ grep “wilco[cx]k*s*” emp.lst
- * has significance in a regular expression only if it is preceded by a character.

# The Dot

- A . matches a single character.
- The .* signifies any number of characters, or none.

  $ grep “p.*woodhouse” emp.lst  (if you want to look for p. woodhouse but not sure whether it actually exists as p.j. woodhouse)

- If look for the name p.j. woodhouse, then the expression should be p\.j\., the dots need to be excaped here.
- To look for a pattern g*, use grep “g*”, to look for a [, use \[, to look for a pattern .* use \.*

# Specifying Pattern Locations (^ and $)

- $ grep “^2” emp.lst
  - locate lines where the emp-id begins with 2.
- To select lines where the salary lies between 70,000 and 89,999:
  - $ grep “[78]....$” emp.lst
- To select only those lines where the emp-ids don’t begin with a 2:
  - $ grep “^[^2]” emp.lst (first ^ anchors the pattern, last ^ negates a class)
- To list only directories:
  - $ ls -l | grep “^d”
- To identify files with specific permissions (write for the group):
  - $ ls -l | grep ‘^.....w’

# Extended Regular Expressions (ERE)

- + matches one or more occurrences of the previous character
- ? matches zero or one occurrence of the previous character
- Need to use option -E: $ grep -E “true?man” emp.lst
- | is the delimiter of multiple patterns:
  - $ grep -E ‘woodhouse|woodcock’ emp.lst
  - $ grep -E ‘wood(house|cock)’ emp.lst
  - $ grep -E ‘wilco[cx]k*s*|wood(house|cock)’ emp.lst

# ERE Set used by grep, sed, and awk

| Expression | Matches |
|---|---|
| *ch*+ | One or more occurrences of character *ch* |
| g+ | At least one g |
| *ch*? | Zero or one occurrence of character *ch* |
| g? | Nothing or one g |
| *exp1*\|*exp2* | Expression *exp1* or *exp2* |
| GIF\|JPEG | GIF or JPEG |
| (*x1*\|*x2*)*x3* | Expression *x1x3* or *x2x3* |
| (lock\|ver)wood | lockwood or verwood |

# sed: The Stream Editor

- Performs non-interactive operations on a data stream.
- Instruction format: sed options ‘address action’ file(s)
- Addressing is done in two ways:
  - By one or two line numbers (like 3,7)
  - By specifying a /-enclosed pattern which occurs in a line (like /From:/)

# Internal Commands Used by sed

| Command | Significance |
|---|---|
| i, a, c | Inserts, appends and changes text |
| d | Deletes line(s) |
| 1,4d | Deletes lines 1 to 4 |
| r foo | Places contents of file foo after line |
| w bar | Writes addressed lines to file bar |
| p | Prints line(s) on standard output |
| 3,$p | Prints lines 3 to end (-n option required) |
| $!p | Prints all lines except last line (-n option required) |
| /begin/,/end/p | Prints lines enclosed between begin and end (-n option required) |
| q | Quits after reading up to addressed line |
| 10q | Quits after reading the first 10 lines |
| = | Prints line number addressed |
| s/*s1*/*s2*/ | Replaces first occurrence of string or regular expression *s1* in all lines with string *s2* |
| 10,20s/-/:/ | Replaces first occurrence of - in lines 10 to 20 with a : |
| s/*s1*/*s2*/g | Replaces all occurrences of string or regular expression *s1* in all lines with string *s2* |
| s/-/:/g | Replaces all occurrences of - in all lines with a : |

# Line Addressing

- $ sed -n '1,2p' emp.lst (use -n to suppress selected lines appear twice behavior of the p command)
- To select lines 9 through 11:
  $ sed -n '9,11p' emp.lst
- To select multiple groups of lines:
  $ sed -n '1,2p
  7,9p
  $p' emp.lst
- Same as above:
  $ sed -n '1,2p;7,9p;$p' emp.lst
- Negating the action (!):
  $ sed -n '3,$!p' emp.lst (Don't print line 3 to the end)

# sed Options

- Multiple Instructions in the Command Line (-e):
  - $ sed -n -e ‘1,2p’ -e ‘7,9p’ -e ‘$p’ emp.lst
- Takes instructions from file filename (-f):
  - $ cat instr.fil

    1,2p

    7,9p

    $p
  - $ sed -n -f instr.fil emp.lst
  - $ sed -n -f instr.fil -f instr.fil2 emp.lst
  - $ sed -n -e ‘/wilcox/p’ -f instr.fil -f instr.fil2 emp?.lst

# Others

- Context Addressing: the pattern has a / on either side:
  - $ sed -n '/From: /p' $HOME/mbox
- Writing selected lines to a file (w):
  - $ sed '/<Form>/,/<\/FORM> /w forms.html' pricelist.html
- You can search for three sets of patterns and store in three files:
  - $ sed -e '/<Form>/,/<\/FORM> /w forms.html
    /<FRAME>/,/<\/FRAME> /w frames.html
    /<TABLE>/,/<\/TABLE> /w tables.html' pricelist.html

# The class is done!

Back to slide 14 to play some games!