Unit 9
Regular Expressions and Text Selection Utilities

Unit Objectives

After completing this unit, you should be able to:

- Use **`regular expressions`**
- Use the **`grep`** command
- Use the **`tr`** command
- Use the **`cut`** command
- Use the **`paste`** command

Sample Data File

To manipulate data, we need to know its format.

The data file we will use in this unit has the following structure:

```
Lastname,<Space>Firstname<Tab>nnn-mmmm
```

```
$ cat  phone.list
Terrell, Terry                617-7989
Franklin, Francis             704-3876
Patterson, Pat                614-6122
Robinson, Robin               411-3745
Christopher, Chris            305-5981
Martin, Marty                 814-5587
Llewellyn, Lynn               316-6221
Jansen, Jan                   903-3333
Llewellyn, Lee                817-8823
$ _
```

Regular Expressions

Powerful feature available in many programs

Used to **select** text in:

– vi, ex, emacs, grep/egrep, sed, awk, perl

What are **RegExes**?

– An expression representing a pattern of characters

– Contain a sequence of characters/metacharacters

Regular Expression Metacharacters

Pattern	Matches
alphanumeric character	The character itself (not really a metacharacter)
. (period)	Any single character
[AZ]	One of A or Z
[^AZ]	Any character not A or Z
[A-Z]	Any character in range A to Z
[-AZ]	One of -, A or Z
[0-9]	Any digit 0 to 9

Extending the Pattern

Two ways:

- Anchors
- Multipliers

Anchors are

^	Matches beginning of line
$	Matches end of line

Multipliers apply to patterns. They are:

*****	zero or more occurrences of previous pattern
?	zero or one occurrence of previous pattern
+	one or more occurrences of previous pattern
{m,n}	at least **m** and no more than **n** occurrences of previous pattern ("quoted braces")

Simple Regular Expression Example

What would the following match?

```
grep '^[M-Z]' phone.list
grep '^[^M-Z]' phone.list
grep '^L.*3$' phone.list
grep '^P.tt' phone.list
grep 'n\*' phone.list
```

Quoted Braces

To specify the number of consecutive occurrences

Syntax 1: `regular_expression\{min, max\}`

To look for two, three or four occurrences of any combination of the characters 3, 4 and 5 consecutively

```
grep '[345]\{2,4\}' phone.list
```

Syntax 2: `regular_expression\{exact\}`

To look for any lines which have two consecutive "**r**" characters

```
grep 'r\{2\}' phone.list
```

Syntax 3: `regular_expression\{min,\}`

To look for any lines with at least two consecutive "**r**" characters preceded by an "e"

```
grep 'er\{2,\}' phone.list
```

Quoted Parentheses

To capture the result of a pattern:

Syntax: `\(regular expression\)`

- Stores the character(s) that match the regular expression (within parentheses) in a register.
- Nine registers are available; characters which match the first quoted parentheses are stored in register one, those that match the second quoted parentheses in register two, and so forth.
- To reference a register use a backslash followed by a register number:

  `\1` to `\9`

For example, to list any lines in "phone.list" where there are two identical characters together...

```
grep  '\(.\)\1'  phone.list
```

Regular Expressions – Quiz

Using the "phone.list" file, what RE gives:

1. People with six-letter surnames?

2. People with first names of at least four characters?

3. All entries where the number before the dash is the same as that after the dash for example 3-3456?

4. People whose surnames begin with A, B or C?

grep Command

- Search files or standard input for lines containing a match for a specific pattern

```
grep [options] pattern [ file1 file2 . . . ]
```

- Valid options:

`-c`	print only a ***count*** of matching lines
`-i`	***ignore*** the case of letters when making comparisons
`-l`	***list*** only the names of the files with matching lines
`-n`	***number*** the matching lines
`-s`	works ***silent***ly, does not display error messages
`-v`	print lines that do NOT match
`-w`	do a whole ***word*** search

grep Examples

```
$ grep -i "tech support" phone.list

$ grep bob /etc/passwd

$ ps -ef | grep chris

$ ls -l | grep '^d'

$ grep -n '.*' /etc/passwd > \

> passwd.file.numbered.lines

$ egrep 'gene|jean' /etc/passwd
```

tr For Translations

The **tr** command translates one set of characters into another:

```
tr LISTIN LISTOUT < in_file > out_file

  – or -

tr -d LISTIN < in_file > out_file
```

- Characters in **LISTIN** are replaced by the corresponding ones in **LISTOUT**
- If **LISTOUT** contains fewer characters than **LISTIN** ignores extra ones from **LISTIN**
- If **LISTOUT** contains more characters than **LISTIN** ignores extra ones from **LISTOUT**
- With **-d**, characters in **LISTIN** are deleted
- Only works with **STDIN** and **STDOUT**
- The **-s** option squeezes multiple characters in a row into one character

tr Examples

Some simple translations...

```
$ print $HOME | tr "/" "-"
-home-team01
$ print "{ { [ ... ] } }" | tr "{}" "()"
( ( [ ... ] ) )
$ print "Lower to upper" | tr "[a-z]" "[A-Z]"
LOWER TO UPPER
$ print "TOP DOWN" | tr '[:upper:]' '[:lower:]'
top down
$ print "vowels and consonants" | tr -d 'aeiou'
vwls nd cnsnnts
$ tr -d '\015' < dos_txt_file > aix_txt_file
$ print 'Lynn Llewellynn' | tr -s "ln"
Lyn Lewelyn
$ _
```

The cut Command

cut extracts fields or columns from text input

```
cut -dS -s -flist [ file ]

        or

cut -cLIST [ file ]
```

-dS	where **S** is the character to take as a delimiter (**<Tab>** ***is default***)
-s	with **-d s** suppresses lines that do not contain delimiters
-fLIST	specifies a **LIST** of fields to cut out and keep
-cLIST	is a **LIST** of columns to cut (character positions)
LIST	- specifies field or column numbers - may contain comma separated values **(m,n)** or a range **(m-n)**

cut Examples

Field numbering starts at 1

```
$ cut  -d:  -f1,3,4  /etc/passwd  |  head  -3
root:0:0
daemon:1:1
bin:2:2
...
robin:0:0
$ _
$ df | cut  -c-12,35-41 | sort
/dev/hd1              4%
/dev/hd10opt         55%
/dev/hd2             95%
/dev/hd3              6%
/dev/hd4             39%
...
$ _
```

What could this mean?

What If There Is No Common Delimiter?

- Using **`tr -s`** and **`cut -d`**, have the output from the **`df`** command only show **`%used`** and **`mount point`**

- Using only **`cut -c`**, have the output from the **`df`** command only show **`%used`** and **`mount point`**

- We will do this again later using **`awk`**

The paste Command

As name suggests, **`paste`** sticks or merges things together

Commonly used to create or format a data stream

Default output is
  line from file1 **`<Tab>`** line from file2

Separators may be changed on command line

Options:
- **`-d [dlist]`** the delimiter between files (may be a list)
- **`-s`** make the output a single line in each file

Checkpoint

1. What **`Reg Ex`** can you use to select surnames?
2. What **`regular expression`** can you use to select text with repeated characters in the surname?
3. What command can you use to select lines in phone.list with four character first names?
4. How could you count the number of processes whose PIDs are in the range 1000-9999?
5. How would you convert spaces to a tab in phone.list?
6. What would this next command accomplish?

```
cut -d: -f1,3,4 /etc/passwd
```

7. Using the **`paste`** command, output the /etc/passwd file so that each line of information is separated by a tab and so that the fifth, sixth and seventh fields are on a separate line from the others. (Hint: make each field a line.)

Unit Summary

- Understanding **regular expressions**
- Using the **grep** command to select text
- Using the **tr** command to translate characters
- Using the **cut** command to select text fields
- Using the **paste** command to merge data streams