Regular expressions

- Key to powerful, efficient, and flexible text processing
- Defined as a string composed of letters, numbers, and special symbols, that defines one or more strings
- You have already used them in selecting files when you used asterisk (`*`) and question mark characters to select filenames
- Used by several Unix utilities such as **`ed`**, **`vi`**, **`emacs`**, **`grep`**, **`sed`**, and **`awk`** to search for and replace strings
  - Checking the author, subject, and date of each message in a given mail folder

```
egrep "^(From|Subject|Date): " <folder>
```

  - The quotes above are not a part of the regular expression but are needed by the command shell

- A regular expression is composed of characters, delimiters, simple strings, special characters, and other metacharacters defined below
- Characters
  - A character is any character on the keyboard except the newline character '`\n`'
  - Most characters represent themselves within a regular expression
  - All the characters that represent themselves are called *literals*
  - A special character is one that does not represent itself (such as a metacharacter) and needs to be *quoted*
    - The metacharacters in the example above (with **`egrep`**) are `"`, `^`, `(`, `|`, and `)`
  - We can treat the regular expressions as a language in which the literal characters are the *words* and the metacharacters are the *grammar*

- Delimiters
  - A delimiter is a character to mark the beginning and end of a regular expression
  - Delimiter is always a special character for the regular expression being delimited
  - The delimiter does not represent itself but marks the beginning and end of the regular expression
  - Any character can be used as a delimiter as long as it (the same character) appears at both ends of the regular expression
  - More often than not, people use forward slash '`/`' as the delimiter (guess why)
  - If the second delimiter is to be immediately followed by a carriage return, it may be omitted
  - Delimiters are not used with the **`grep`** family of utilities

- The metacharacters in the regular expressions are

```
^  $  .  *  [  ]  \{  \}  \  \(  \)
```

  - In addition, the following metacharacters have been added to the above for extended regular expressions (such as the one used by **`egrep`**)

    ```
    +  ?  |  (  )
    ```

  - The dash (`-`) is considered to be a metacharacter only within the square brackets to indicate a range; otherwise, it is treated as a literal
    - ∗ Even in this case, the dash cannot be the first character and must be enclosed between the beginning and the end of range characters
- The regular expression search is not done on a word basis but utilities like **`egrep`** display the entire line in which the regular expression matches

- Simple strings
  - The most basic regular expression
  - Matches only itself
  - Examples

Reg. Exp.	Matches	Examples
/ring/	ring	ring
		spring
		ringing
		stringing
/Thursday/	Thursday	Thursday
		Thursday’s
/or not/	or not	or not
		poor nothing

- Special characters
  - Cause a regular expression to match more than one string
  - Period
    * Matches any character
    * Examples

Reg. Exp.	Matches	Examples
/ .alk/	All strings that contain a space followed by any character followed by alk	will talk may balk
/.ing/	all strings with any character preceding ing	singing ping before inglenook

/09.17.98/	Date with any separator	`09/17/98`
		`09-17-98`

- Square brackets
  - Define a class of characters that matches any single character within the brackets
  - If the first character immediately following the left square bracket is a caret '`^`', the square brackets define a character class that match any single character not within the brackets
  - A hyphen can be used to indicate a range of characters
  - Within a character class definition, the special characters (backslash, asterisk, and dollar signs) lose their special meaning
  - A right square bracket appearing as a member of the character class can only appear as the first character following the square bracket
  - A caret is special only if it is the first character following the square bracket
  - A dot within square brackets will not be a metacharacter
    - `/07[.-]17[.-]98/` will not match `07/17/98` but will match `07-17-98`

* Examples

Reg. Exp.	Matches	Examples
/[bB]ill/	Member of the character class b and B followed by ill	bill Bill billed
/t[aeiou].k/	t followed by a lowercase vowel, any character, and a k	talkative stink teak tanker
/number [6-9]/	number followed by a space and a member of the character class 6 through 9	number 60 number 8: get number 9

`/[^a-zA-Z]/`	any character that is not a letter	1
		7
		`@`
		`.`
		`}`
		`Stop!`

– Asterisk

  ∗ Can follow a regular expression that represents a single character
  ∗ Represents zero or more occurrences of a match of the regular expression
  ∗ An asterisk following a period matches any string of characters
  ∗ A character class definition followed by an asterisk matches any string of characters that are members of the character class
  ∗ A regular expression that includes a special character always matches the longest possible string, starting as far toward the beginning (left) of the line as possible

∗ Examples

Reg. Exp.	Matches	Examples
`/ab*c/`	a followed by zero or more b's followed by a c	ac abc abbc debbcaabbbc
`/ab.*c/`	ab followed by zero or more other characters followed by a c	abc abxc ab45c xab 756.345 x cat
`/t.*ing/`	t followed by zero or more characters followed by ing	thing ting I thought of going
`/[a-zA-Z ]*/`	a string composed only of letters and spaces	1. any string without numbers or punctuation!
`/(.*)/`	as long a string as possible between (and)	Get (this) and (that);
`/([^)]*)/`	the shortest string possible that starts with (and ends with)	(this) Get (this and that)

- Caret and dollar sign
  - A regular expression beginning with a caret ‘**`^`**’ can match a string only at the beginning of a line
    - The regular expression **`cat`** finds the string **`cat`** anywhere on the line but **`^cat`** matches only if the string **`cat`** occurs at the beginning of the line
    - **`^`** is used to ***anchor*** the match to the start of the line
  - A dollar sign ‘**`$`**’ at the end of a regular expression matches the end of a line
    - The regular expression **`cat`** finds the string **`cat`** anywhere on the line but **`cat$`** matches only if the string **`cat`** occurs at the end of the line, it cannot be followed by any character but newline (not even space)

∗ Examples

Reg. Exp.	Matches	Examples
`/^T/`	a T at the beginning of a line	This line ... That time...
`/^+[0-9]/`	a plus sign followed by a number at the beginning of a line	+5 + 45.72 +759 Keep this...
`/:$/`	a colon that ends a line	...below:

– Quoting special characters

* Any special character, except a digit or a parenthesis, can be quoted by preceding it with a backslash
* Quoting a special character makes it represent itself
* Examples

Reg. Exp.	Matches	Examples
`/end\./`	all strings that contain end followed by a period	The end. send. pretend.mail
`/\\/`	a single backslash	`\`

`/\*/`	an asterisk	`*.c`
		`an asterisk (*)`

`/\[5\]/`	`[5]`	it was five [5]
`/and\/or/`	`and/or`	`and/or`

- Rules
  - Longest match possible
    - * A regular expression always matches the longest possible string, starting as far towards the beginning of the line as possible
  - Empty regular expressions
    - * An empty regular expression always represents the last regular expression used
    - * Let us give the following command to `vi`

```
:s/mike/robert/
```

    - * If you want to make the same substitution again, the following is sufficient

```
:s//robert/
```

    - * You can also do the following

```
/mike/
:s//robert
```

- Bracketing expressions
  - Regular expressions can be bracketed by quoted parentheses `\(` and `\)`
  - The string matching the bracketed regular expression can be subsequently used as quoted digits
  - The regular expression does not attempt to match quoted parentheses
  - A regular expression within the quoted parentheses matches exactly with what the regular expression without the quoted parentheses will match
  - The expressions **`/\(rexp\)/`** and **`/rexp/`** match the same patterns

- Quoted digits
  * Within the regular expression, a quoted digit (`\n`) takes on the value of the string that the regular expression beginning with the nth `\(` matched
  * Assume a list of people in the format

    last-name, first-name initial
  * It can be changed to the format

    first-name initial last-name

    by the following `vi` command

```
:%s/\([^,]*\), \(.*\)/\2 \1/
```

- Quoted parentheses can be nested
  * There is no ambiguity in identifying the nested quoted parentheses as they are identified by the opening `\(`
  * Example

```
/\([a-z]\([A-Z]*\)x\)
```

    matches

```
3 t dMNORx7 l u
```

- Replacement string
  - **`vi`** and **`sed`** use regular expressions as search strings with the substitute command
  - Ampersands (**`&`**) and quoted digits (**`\n`**) can be used to match the replacement strings within the replacement string
  - An ampersand takes on the value of the string that the search string matched
  - Example

```
:s/[0-9][0-9]*/Number &/
```

- Word boundaries
  - The word boundaries in the regular expressions are denoted by any whitespace character, period, end-of-line, or beginning of line
  - Expressed by

```
\<  beginning of word
\>  end of word
```

- Regular expressions cannot be used for the newline character