**Practical computing for biologists**

**I) Ch.1**

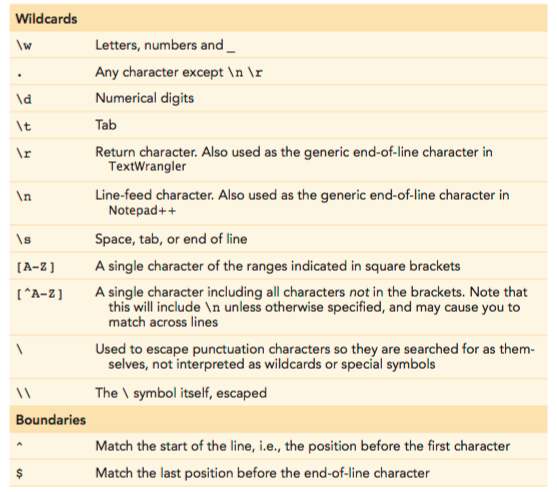
TextWrangler for text editing.

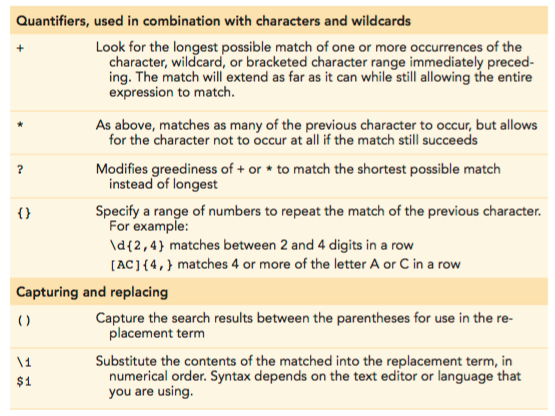
Line ending: what is used to pass from one line to another (↵enter). It is not standardized and can cause problems. There are two: **carriage return** (CR or \r) and **line feed** (LF or \n). To find out which one a document has, open it in textwrangler and check at the bottom. It can be changed.

Macintosh HD:Users:dundon:Desktop:Screen Shot 2016-05-11 at 16.17.10.png

**II) Ch.2**

Regular expressions (regexp, regex, **grep**): can do anything that a simpler search and replace tool does. It can employ **wildcards** (like other search & replace tools), which are characters that can match more than one character (e.g. \* to match as manx characters as possible). But it is more powerful and allows, for example, to use only part of the search term and use it to replace (e.g. with many measurements, you can extract any digits preceeding “cm” and insert them somewhere else).





Search function should be case sensitive and allow Grep.

*Example*:

Agalma elegans

Turn to: Agalama elegans A. elegans

Search: (\w)(\w+) (\w+)

Replace: \1\2 \3 \1. \3

Here we use () and + to give a special meaning to our search. But if we want to search for an actual () or + in our text, we have to devoid it from its special meaning. This is done by placing a \ before the character. To devoid \ of its special meaning we would write \\.

*Example*:

Physalia physalia (Linnaeus)

Turn to: P. physalia (L.)

Search: (\w)\w+ (\w+) \((\w)\w+\) OR (\w)\w+ (\w+) (\()(\w)\w+(\))

Replace: \1. \2 \(\3.\) OR \1. \2 \3\4\5

*Example:*

-91 59.8'S -157 58.2'W

Turn to: -91 59.8 -157 58.2 (tabs between numbers: \t)

Search: (.+) (.+)\'\w\s(.+) (.+)\'\w BUT in Sublime Text the tick mark doesn’t need to be escaped: (.+) (.+)'\w\s(.+) (.+)'\w. is used to replace any letter, number or symbol, except end-of-line characters

\s is a space, including tabs and end-of-line

Replace: \1\t\2\t\3\t\4\t

This search term is very general, though, and can be made more specific

Search: (.\d+) (\d+\.\d)\'\w\s(.\d+) (\d+\.\d)\'\w

\d is used for any digit. Note that for decimals, the decimal point and the digits on both side of it have to be written as three entities: 59.8 = \d+\.\d

|  |  |
| --- | --- |
| *Workflow to write a search-replace expression* | |
| Original text | +38 30.5’N |
| Mark captures | (+38) (30.5)’N |
| Regular expression with Wildcards | (.\d+) (\d+\.\d)\’\w |
| Replacement | \1\t\2\t |

By default, TextWrangler and other text editors search line by line. If we want to search across lines, we have to include a end-of-line character at the end of our search term (\r or \n). If we want to preserve this as different lines, we would have to add \r at the end of the replacement term.

*Example:*

-9 59.8'S -157 58.2'W

+21 17.4'N -157 51.6'W

Turn to: -9 59.8 -157 58.2 +21 17.4 -157 51.6

Search: (.\d+) (\d+\.\d)\'\w\s(.\d+) (\d+\.\d)\'\w\r

Replace: \1\t\2\t\3\t\4\t

If we want the result to be in two lines:

-9 59.8 -157 58.2

+21 17.4 -157 51.6

Then the search term is the same but we add an \r at the end of the replace term:

\1\t\2\t\3\t\4\t\r

*Exercise:*

>CAA58790.1= green fluorescent protein [Aequorea victoria]

MSKGEELFTGVVPILVELDGDVNGQKFSVRGEGEGDATYGKLTLKFICTTGKLPVPWPTLVTTFSYGVQCFSRYPDHMKQHDFLKSAMPEGYVQERTIFYKDDGNYKTRAEVKFEGDTLVNRIELKGIDFKEDGNILGHKMEYNYNSHNVYIMGDKPKNGIKVNFKIRHNIKDGSVQLADHYQQNTPIGDGPVLLPDNHYLSTQSALSQDPHGKRDHMVLLEFVTSAGITHGMDELYK

Transform into:

>CAA58790\_Avi

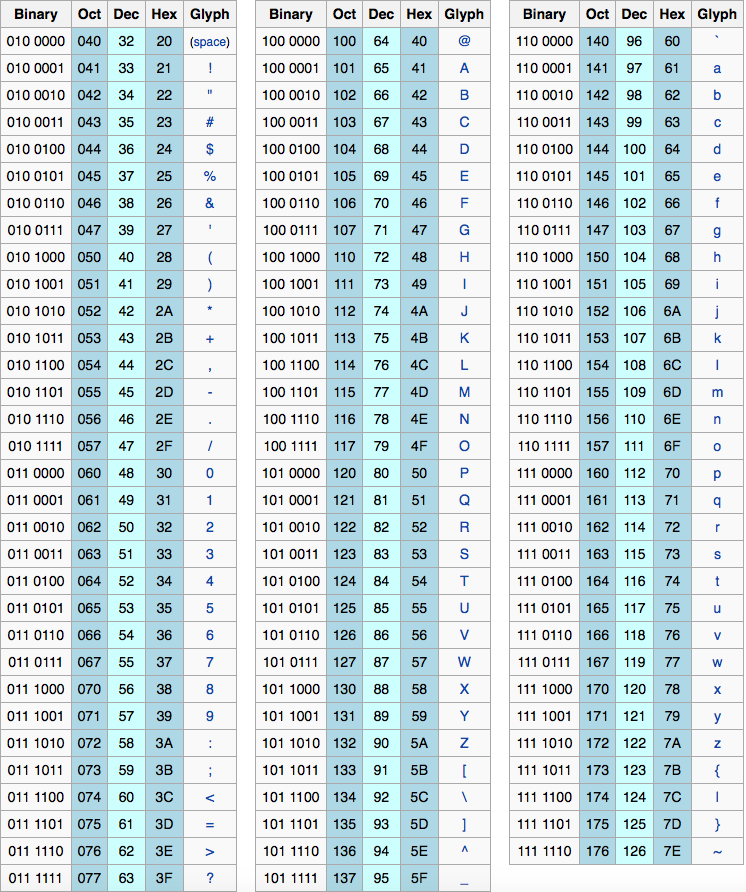
MSKGEELFTGVVPILVELDGDVNGQKFSVRGEGEGDATYGKLTLKFICTTGKLPV…

|  |  |
| --- | --- |
| Original text | >CAA58790.1= green fluorescent protein [Aequorea victoria] |
| Mark captures | (>CAA58790).1= green fluorescent protein [(A)equorea (vi)ctoria] |
| Regular expression with Wildcards | (.+)\..+\[(\w)\w+\s(\w\w)\w+\] |
| Replacement | \1\_\2\3 |

|  |  |
| --- | --- |
| Alternatives | |
| (.+)\..+\[(\w)\w+\s(\w\w)\w+\] | original |
| (.+)\..+\[(\w)\w+\s(\w\w).+ | of course |
| (>\w+).+\[(\w)\w+\s(\w\w).+ | \w is only letters, numbers and \_ so it will automatically end at the period |
| (\>\w+).+\[(\w)\w+\s(\w\w).+ | it’s not necessary to escape > with a \ before, but it’s a good idea to always do it with punctuation marks because they might have a special meaning |

**III) Ch.3**

One can form their own wildcards using [ ]. This will follow the ASCII character set.



[A-z] would replace any single character between A and z. Note that this includes six punctuation marks (i.e. [, \, ], ^,\_ and ‘). If we wanted only letters both in upper- and lowercase, we would have to write [A-Za-z]. As with normal wildcards, a + can be added at the end.

Sometimes it is easier to indicate what an expression does not contain that what it does. In this case we use **[^]** which will select **everything except** what follows the ^. For example, [^A-Z] will select everything except any capital letter A to Z. This is quite useful when, for example, you have a dataset consisting of columns separated by columns. For each line, if you want to select the data from each column but not the tabs that separate the columns, you could use ([^\t]+)\t which means: everything except a tab followed by a tab. This is called **parsing**. Notice that this expression will also include end-of-line characters!

Outside of brakets, **^** indicates the **beginning** of the line (not the first character in the line, but the beginning). Contrarily, **$** indicates the **end** of the line (not the end-of-line character (\n) but the end). So for example, if you want to add a column at the beginning of your dataset, you can search for ^and replace it with Column1\t.

To turn: Agalma elegans

Into: A. elegans

Search: ^(\w)\w+

Replace: \1\.

The search term (\w)\w+ will not include the second word because it is stuck to the beginning of the line by ^. If we took the ^away, it would look for any expression in the line that contains one letter followed by one or more letters, regardless of the position within the line.

+ means that it is present one or more times. **\***, by contrast, means that it is present **zero or more times**. This is critical if you are unsure of whether the character will be present in every case or not.

+ and \* will always try to get as many matches as possible, so they will search the maximal number of characters that conform to the expression preceeding it; in other words, they are **greedy**. By adding **?** after + or \* we will search for the minimun number of characters that conform to it.

*Example*

Turn: CCAGTGCAGTGCGAGAGACTCGAGCTGCAGCAGTCGTGAAAAAAAAAAAAAA

Into: CCAGTGCAGTGCGAGAGACTCGAGCTGCAGCAGTCGTG

In other words, eliminate the poly-A tail.

Search: (\w+)A\*

The term A\* won’t find anything, because + is greedy and it will search for the maximum number of word characters and include them in the parenthesis! It’s greedy

Search: (\w+)[CTG]A\*

Replace: \1

Result: CCAGTGCAGTGCGAGAGACTCGAGCTGCAGCAGTCGT

This will work because (\w+)[CTG] will search for the maximum number of consecutive characters whose last member is a C, T or G. If there are ny missing characters, or for example n, it might not work if the n is present before the poly-A.

Search: (\w+?)A+

This will only choose the first three characters (CCA), because it will include the minimal set of characters before an A (plus the A). Note that replacing the + for a \* will choose individual characters or individual characters plus all the As that follow it.

Search: (\w+?)A\*$ (or +instead of \*)

Replace: \1

Result: CCAGTGCAGTGCGAGAnGACTCGAGCTGCAGCAGTCGTG

This works because it will choose the minimal set of characters that precede a series of As at the end of the line.

+ for 1 or more, \* for 0 or more. For finer control, use {}. {3} after a search term means that it must be matched 3 times. {2,5}, that it should be matched at least 2 times but not more than 5. {3,} means at least 3.

*Example*

13.283764872634876234982734

Turn to: 13.283

Search: (\d+.)(\d{3})\d+

Replace: \1\2

Notice that it will not round up the last digit!

*Example*

13 January, 1752 at 13:53 -1.414 5.781 Found in tide pools

17 March, 1961 at 03:46 14 3.6 Thirty specimens observed

1 Oct., 2002 at 18:22 36.51 -3.4221 Genome sequenced to confirm

20 July, 1863 at 12:02 1.74 133 Article in Harper's

Rearrange data into 7 columns:

Year Mon. Day Hour Minute X data Y data

|  |  |
| --- | --- |
| Original text | 13 January, 1752 at 13:53 -1.414 5.781 Found in tide pools |
| Mark captures | (13) (Jan)uary, (1752) at (13):(53) (-1.414) (5.781) Found in tide pools |
| Regular expression with Wildcards | (\d+)\s(\w{3}).\*\s(\d{4}).+(\d\d)\:(\d\d)\t(.+)\t(.+)\t.+  or  ^(\d+) (\w\w\w).\*, (\d+) at (\d+):(\d+)\s+([\d\.\]+)\s+([\d\.\-]+).+ |
| Replacement | \3\t\2.\t\1\t\4\t\5\t\6\t\7 |

Note to self: use more customized regular expression, even if just as an exercise.

**Ch. 4 – Command-line operations: the shell**

Graphical user interfaces **(GUIs)** are the interfaces with which we normally interact with the computer. They are frendly and intuitive. **Command line** is another way. The interaction is mediated by the Shell, which in Mac is called Terminal.

The description of where something on the computer is located is called a path. A path can be absolute (/Users/David/Documents/doc.txt). Slashes are used to separate directories, but if it’s at the beginning, it means the root directory. Paths can also be relative, for which you need to know your reference point. If you are in /Users/David/, then a relative path would be Documents/doc.txt. Note that it does not begin with a backslash, because that would mean the root.

Command line is the set of instructions that will transform, for example, the raw data into a graph.

We open terminal to carry out commands (like cmd in windows): applications-utilities-terminal

ls and pwd are examples of programs that are carried out by the terminal. We can modify these programs to fit our need by using arguments. For example, the argument –l will modify ls so that it shows additional information about the files that are in the (print work directory).