Latin Word Study with Notepad++

Index

Introduction

This short tutorial will show you how to use the Windows text editor *Notepad++*[1] to search multiple text files for occurrences of a search pattern (irrespective of language of the text, and therefore also suitable for Latin) in order to study the use of specific Latin words. Its purpose is not only to show how to use the programme, but especially to show the usefulness of so-called „regular expressions" (abbreviated here as RegEx). This tutorial does *not* explain how regular expressions work, it only strives to show why they are so useful. Links to RegEx-tutorials are given at the end of the tutorial.

What you need

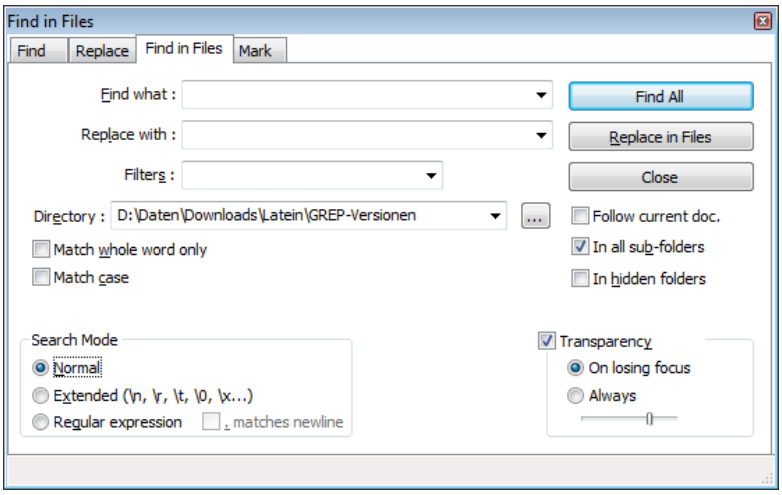
- *Notepad++*: „Notepad++ is a free (as in ‚free speech' and also as in ‚free beer') source code editor and Notepad replacement that supports several languages."[2]
- Several plain text files, which excludes RTF and DOC-files, but does include HTML-files (although it is best to strip these of all tags to make them more legible). These files need to be stored in a common directory (or in a sub-directory of a common directory). For this tutorial we shall use two Latin books transcribed by me, both of which can be downloaded from Gutenberg.org, and one of Livy's books from The Latin Library:
  - „Mysterium Arcae Boulé" by Burton E. Stevenson (translated by Arcadius Avellanus)[3]
  - „Pericla Navarchi Magonis" by Léon Cahun (translated by Arcadius Avellanus)[4]
  - „Ab urbe condita, liber XXXIII" by Titus Livius.[5] Do not save this page as a HTML-file but as a text-file so that you get rid of the HTML-tags.

[1] The examples were created using Notepad++ v6.7.7 (build time: 16 April 2015).
[2] Source: http://notepad-plus-plus.org/ (Retrieved: 2015-05-05)
[3] http://www.gutenberg.org/ebooks/46456
[4] http://www.gutenberg.org/ebooks/48681
[5] http://www.thelatinlibrary.com/livy/liv.33.shtml

Let's begin

First start *Notepad++*.

Then open the „Find in Files"-dialogue. To do so, press <Ctrl>-f. This will open the „find"-window which features four tabs: „Find", „Replace", „Find in Files", and „Mark." Select the tab „Find in Files." You will be presented with a dialogue that looks like this:



It can be immediately seen that this dialogue is used both for „finding" and „finding and replacing." Therefore it is advised to make sure that you indeed press „Find All" later on and NOT „Replace in Files" unless you want the text files to be altered. Now press on the button „..." beside the „Directory"-box and find and select the directory where you have stored the three text files. The field „Filters" can be used to select a certain range of files in the selected directory. Choose „*.*" if you want to apply the search filter to all files, or „*.txt" if only files with the extension „.txt" are to be examined.

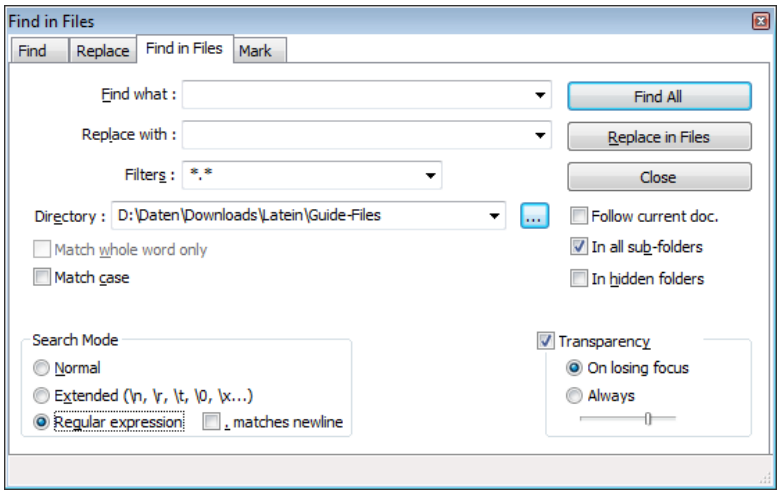
The option „In all sub-folders" needs to be checked if your text files are distributed among one or more sub-directories. The „Match case"-option should be unchecked. Otherwise you need to know exactly whether the letters of your search pattern are capitalized or not. You can ignore „Transparency". The option „Match whole word only" can be checked or unchecked depending on what you are exactly looking for. It is not available in the search mode „Regular expression."

Finally you have to select the „Search Mode". There are three modes:

- „Normal": this mode can be used whenever you know exactly what you are looking for. You might use this option to search for occurrences of the word „iterum". It will find both „iterum" and „iterumque".[6] Should you be interested solely in „iterum", then you need to check the option „Match whole word only".
- „Extended": you can use this mode if special characters like line feed, tabulator, etc. need to be included in the search expression. This mode is the least useful one for studying the use of words (but very helpful for other tasks).
- „Regular expression": by far the most powerful mode, it also requires the most knowledge. However, it is definitely worth learning at least the basics of regular expressions. All examples shown will use them. So check this mode.

Once you have done all of this, the „Find in files" dialogue should look like this:

[6] Actually, it finds the „iterum" in „iterumque".



Now we are ready for some examples.

Examples

Example 1: use of the verb „applicare“

Latin is a highly inflected language with various word endings which means that the word might be „applicare“, „applicat“, „applicuit“, etc. There is another complication, however. This word has an alternative form „adplicare“ with the resulting forms „adplicat“, „adplicuit“, etc. One could, of course, make seperate search runs for all of these forms, but regular expressions allow to find all at once. So, let's have a go at it (make sure that „Regular expression“ is the selected search mode). Here, and in the following examples, type the search pattern into the field „Find what“.

Search pattern: \ba[dp]plic\w*\b

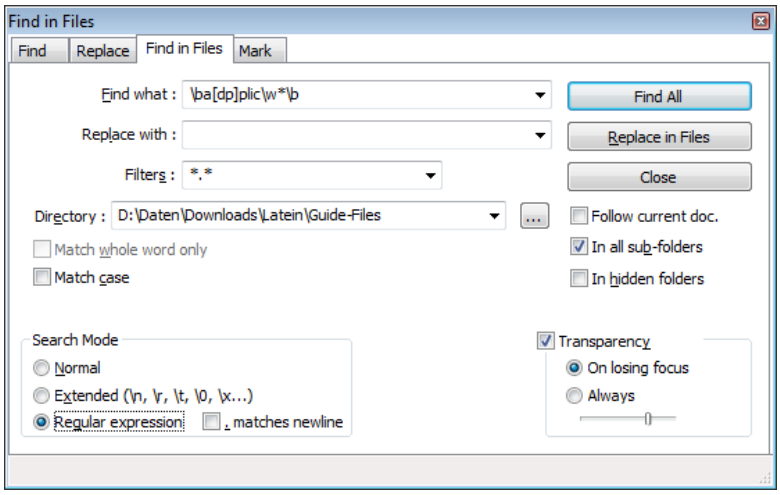
At the beginning I wrote that this is not a RegEx-tutorial, but we shall nonetheless have a look at what the above means.

The leading and trailing „\b“ mark wound boundaries (beginning and end of a word) because we want to find only complete words (this replaces the option „Match whole word only“ available for the other two search modes). The „a“ is simply the leading „a“ in the word „applicare“ (or „adplicare“). What about „[dp]“? This is a character class (or character set) meaning that we are looking for *either* a „d“ *or* a „p“. The next characters in the search expression, „plic“, are simply the next ones in the word we are looking for. Then we stumble upon „\w“. This is a shorthand character class encompassing word characters. It therefore stands for all letters (a-z, A-Z).[7] A single „\w“ stands for a single word character. But here it is immediately followed by the character „*“ which has a special meaning. In other contexts[8] the „*“ is used as a wildcard character for any number of any character. In the context of regular expressions, on the contrary, the „*“ means „repeat the preceding token zero or more times“. So, „\w*“ stands in for zero or more word characters: for „at“ in „applicat“, „uit“ in „applicuit“, etc.

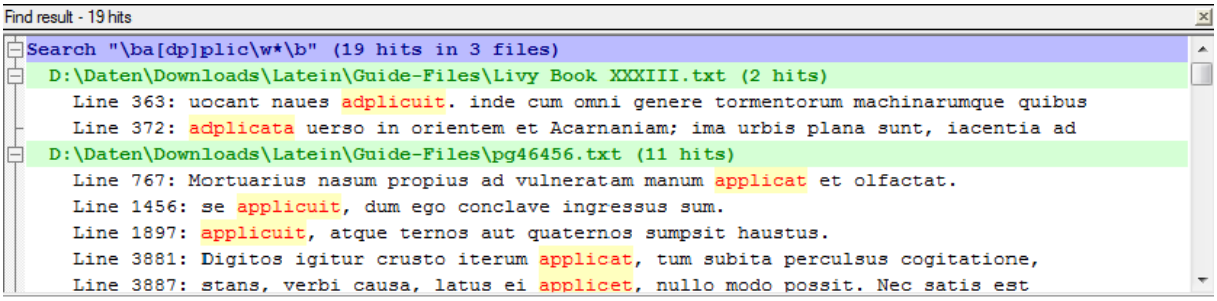
Now, let's have a look at what Notepad++ can do for us. Your dialogue should look like this:

[7] And digits and underscores. For more, see one of the RegEx-tutorials mentioned at the end.

[8] Not regular expressions.



Press the button „Find All“. The programme quickly searches through the selected files and indeed finds „19 hits in 3 files.“ A list of the found occurrences is displayed in a message window:



In this message window the files (green) and the „hits“ (with preceding line numbers) in each of these are displayed. Each match of the search expression is highlighted. But there is more. Move the mouse course on top of one of the lines with a match and double click. Immediately the respective file is opened in the text editor and the cursor jumps to the match. Choosing another match of the same file in the message window will jump to the respective line in the text file. Select a match of a different file, and that is opened as well, without the previously opened text files being closed.

Now click inside the message window to give it the focus, then press Ctrl+A to select all the text, and Ctrl-c to copy it into the clipboard. Next open a new text editor window (Ctrl-n) and insert the text using Ctrl-v. Voilà, the result of your „Find in files“-search is ready for you to analyze, edit and save.

Example 2: use of the word „vires“

Search pattern: \bvires\b

Type the search pattern into the „Find what“-field and press „Find next.“ Now let us have a look at the message window. The result of our latest search is being displayed, but if you scroll to the bottom of the message window, then you will see that the results of the previous „applicare“-related search have been preserved, albeit collapsed so that the result does not take up too much space. If it gets too confusing after a few searches, you can delete the contents of the message windows by selecting everything (Ctrl-a) and deleting it.[9]

Now, let's have a look at the result of our search: „14 hits in 2 files“. Wonderful, but wait...we have 3 files: two translations by Arcadius Avellanus and Livy's text. There are only results from the modern texts, however, not from Livy. Did he really not use „vires“, at least

[9] You have to do this once for each search.

a single time? He did, but the occurrences were not found using the above search pattern because the „Latin Library"-edition of this Livy-text does not use the letter „v" so that „vires" becomes „uires". This gives us a wonderful occasion for using character classes (cp. Example 1). We change the search pattern to:

Search pattern: \b[uv]ires\b

We start the search and get the followig result: „18 hits in 3 files". Good. But still not perfect, there is still one match missing. The reason for this is the enclitic „-que" sometimes tacked on Latin words. How can we deal with these? Try this pattern:

Search pattern: \b[uv]ires(que)?\b

The result: „19 hits in 3 files," and the missing match was „viresque". We dealt with the enclitic by using „(que)?" The round brackets mark „que" as a „capturing group". Capturing groups are used to apply a quantifier (e.g. „*" from Example 1) not to a single letter but to a group of letters (among other things). In this example, it allows the quantifier „?" to be applied to „que". The „?" tells the search engine to match „que" once or not at all making it optional.

All in all, the last search expression finds: „vires", „viresque", „uires", „uiresque".

Example 3: use of words for „door."

The two most important words for „door" are „ianua" and „ostium."

Search pattern: \b(ianua|ostium)\b

The round bracket again acts as a capturing group, in this case, however, for „alternation". Alternation is used to match either[10] of several patterns, here between „ianua" and „ostium". This gives us only the nominative singular of „ianua" and the nominative and accusative singular of „ostium" („52 hits in 2 files"). We want all cases, however. So:

Search pattern: \b(ianu|osti)\w*\b

Now we are presented with „167 hits in 2 files." In this new pattern we put only those parts of the words into a capturing group for alternation which are the same in all cases. The „\w*" will account for the rest of the word.

Why again „in 2 files" instead of in all three? Again there was no match in the Livy-text file. In this case the answer is simply that Livy used neither „ianua" nor „ostium". But we do encounter „doors" of some sort, namely gates. To include these, we again change the pattern:

Search pattern: \b(ianu|osti|port)\w*\b

The count rises to „302 hits in 3 files." Is that all we can learn about „doors" in these texts? No. We are still missing the matches from the word „foris" (often in plural). Incorporating this into the search pattern is a bit more complicated. We could simply add the constant part of the word („for") to the search pattern:

Search pattern: \b(ianu|osti|port|for)\w*\b

But now we are confronted with a staggering „1075 hits in 3 files." What is going wrong? The answer is that we are now finding all (!) words beginning with „for", which includes „forte", „fortuna", „foret", „foro", „fornices", „fornicatu", etc. One way to solve this conundrum is stacking the capturing groups:

Search pattern: \b(((ianu|osti|port)\w*)|(fores|forium|foribus))\b

This new pattern results in a moderate increase in matches to „330 hits in 3 files." The logic behind this new pattern should be clear: its either (!) a word beginning with „ianu" or „osti"

[10] It acts as an *exclusive* „or".

or „port“ or (!) one of the words „fores“, „forium“, „foribus“.[11] I could have included the singular forms of „foris“ as well, but decided against it because that would include the adverb „foris“. And since the word is mostly used in plural, I concentrate on those forms.

As a last addition we take care of the enclitic „-que“:

Search pattern: \b(((ianu|osti|port)\w*)|(fores|forium|foribus))(que)?\b

Now we are left we „334 hits in 3 files“, the new additions are 4 occurrences of „foresque“.

Having a closer look at the list of matches in the message window reveals that there are quite a view false positives, due mostly to other words beginning with „port“ like „Portianum“, „portátilis“ (note that „á“ is also a word character of the shorthand character class \w). It would be possible to further narrow the search, but you need to consider the tradeoff. Either less „hits“ and a more complicated search pattern (and more opportunity to construct it the wrong way), or more false positives and a regular expression that is easier to understand and create.

Some hints

You can, of course, use these regular expressions not only when searching external text files, but also to look for matches in a text file already present in the text editor. In that case you have to choose the normal „Find“-tab in the „Find“-window, enter your regular expression, select the respective search mode and press the button „Find All in Current Document“.

Sometimes you may have to deal with texts having very long lines. In that case the matches in the message window will look weird because the lines are not wrapped in it. In this case it is best to split the long lines into shorter ones. You can do this in Notepad++ by first selecting the text you want to split and then choosing (from the menu) first „Edit“, then „Line Operations“, and finally „Split Lines“. That will do the trick.

Conclusion

I hope that I was able to show how useful regular expressions are for learning Latin. In the past I was a firm believer in regular expressions being a „cruel and unusual punishment.“ Now, however, I would go so far as to claim that they constitute a part of computer literacy, at least as far as the fundamentals are concerned.

These fundamentals can be acquired in a very short time (perhaps during a single afternoon), provided that you use a good tutorial. The best RegEx-tutorials known to me are:

- Regular-Expressions.info[12]
- RexEgg[13]

A word of advice: make sure that you understand what it means that a regular expression-engine is „greedy“ and „eager.“ You can save yourself a lot of trouble that way.

8 May 2015, Carolus Raeticus

[11] Mind the additional round brackets, they are important.

[12] http://www.regular-expressions.info/ (URL retrieved: 8 May 2015)

[13] http://www.rexegg.com/ (URL retrieved: 8 May 2015)