DDigest (version 0.06)

DDigest (De-Digest) is a program designed to extract individual articles from a digest format mailing list. It will then place these articles into a rnews packet to be imported into an off-line news reader. DDigest will find these digests in either a mail packet (in soup format), or a Yarn folder.

Requirements:

MSDOS or other compatible operating system.

Installation:

Copy ddigest.exe to a directory contained in the path.

Edit a configuration file for each mailing list.

Configuration files:

This contains settings telling DDigest how to find and process a digest. Comments may be placed in the configuration file by placing a "`#`" into the first column.

Ddigest uses regular expressions in searching for the '`FindDigest`', '`FirstAricleBreak`', and '`ArticleBreak`' strings. See the section on regular expressions for additional information.

Finding a digest:

DDigest will look in the "`FindHeader`" of each mail message for the "`FindDigest`" search string. If found, the mail message will be processed as a digest. Otherwise, DDigest will continue looking in the next message. For example, if the configuration file contains the settings:

```
FindHeader=Sender:
FindDigest=yarn-list
```

Any mail message containing the string "`yarn-list`" in the "`Sender:`" header will be

considered a digest.

Finding Articles in a digest:
Once a digest has been found, DDigest will look in the message body for the beginning of the first article using the "`FirstArticleBreak`" setting. The end of each article and the beginning of the next article is found with the "`ArticleBreak`" setting. The extracted article will contain everything from the end of one article break to the beginning of the next article break. The article breaks are not included in any articles.

For example, the digestifier places an introduction followed by a line containing seventy-six dashes at the beginning of the digest. Each article in the digest is separated by three lines, the first one being blank, the second line containing thirty dashes, and the third line blank. The configuration file should contain the settings:

```
FirstArticleBreak=^-{60,100}$
ArticleBreak=^$-{30}$$
```

This will find a line containing between sixty and one hundred dashes, and nothing but dashes, and use it as the break between the introduction and the first article. It will also use any group of three lines where the first and third lines in the group are blank and the middle line contains exactly thirty dashes as a break between articles.

Placing extracted articles into a newsgroup:
If necessary, create the newsgroup in the off-line reader. Yarn will also allow you to set up a posting address so that posted articles will automatically be posted to the list and not to a phantom newsgroup.

DDigest will add a "`Newsgroup:`" line to each extracted article using the "`NewsGroup`" setting in the configuration file. For example, if the configuration file contains the setting:

```
NewsGroup=Yarn-List
```

All extracted articles will be placed into the newsgroup "`Yarn-List`".

Regular Expressions:
Using a regular expression in place of a simple search string will allow searching for a string that will change from one digest to the next. For example, you want to look for the line:

```
This is digest #123 in a series on weather
```

The following regular expression will compare the entire line including the digest number:

```
This is digest #[[:digit:]]+ in a series on weather
```

The '`[[:digit:]]+`' part of the expression will match any sequence of one or more digits. It is composed of an operator '[[:digit:]]', and a modifier '+'. The operator will match one digit. The modifier tells DDigest to apply the operator one or more times. A modifier may follow an operator, ordinary character, or parenthesized expression to match multiple occurrences of what the operator, character, or parenthesized expression would match.

An anchor indicates that the character preceding or following it should start or end a word or line.

To Find	Operator	Examples
Any single character.	`.`	"`f.t`" finds "`fat`," "`fet`," and "`fit.`"
One of the specified characters.	`[]`	"`f[ae]t`" finds "`fat`" and "`fet`," but not "`fit.`"
Any single character except one of the specified characters.	`[^]`	"`f[^ae]t`" finds "`fit`," but not "`fat`" and "`fet.`"
Any word constituent character. (A word constituent character is a letter, number or underline.)	`\w`	"`f\wt`" finds "`fit`," "`fat`," and "`f_t`" but not "`f t`" or "`f,t.`"
Any non word constituent character.	`\W`	"`f\Wt`" finds "`f t`" or "`f,t`" but not "`fit`," "`fat`," and "`f_t.`"
Any character that would otherwise be an operator, modifier, or anchor. (These are "`^.[$()|*+?{\`")	`\`	"`f\+t`" finds "`f+t`" but not "`fft`," while "`f+t`" finds "`fft`" but not "`f+t.`"
A line separator.	`$`	"`f$t`" finds text where "`f`" is the last letter on one line and "`t`" is the first letter on the next line.

Any operator or character could be followed by a modifier to match multiple occurrences of a character or expression.

To Find	modifier	Examples
Zero or more occurrences of the previous character or expression.	`*`	"`f.*t`" finds "`ft`," "`fat`," "`fabt`," and "`f a whole lot more text t.`"
One or more occurrences of the previous character or expression.	`+`	"`f.+t`" finds "`fat`," "`fabt`," and "`f a whole lot more text t`," but not "`ft.`"

Zero or one occurrence of the previous character or expression.	`?`	"`f.*t`" finds "`ft`" and "`fat`," but not "`fabt`" or "`f a whole lot more text t`."
Exactly <n> occurrences of the previous character or expression.	`{n}`	"`fa{3}t`" finds "`faaat`" but not "`faat`" or "`faaaat`."
At least <n> occurrences of the previous character or expression.	`{n,}`	"`fa{3,}t`" finds "`faaat`" and "`faaaat`" but not "`fat`."
Between zero and <n> occurrences of the previous character or expression.	`{,n}`	"`fa{,3}t`" finds "`ft`," and "`faaat`" but not "`faaaat`."
Between <n1> and <n2> occurrences of the previous character or expression.	`{n1,n2}`	"`fa{2,3}t`" finds "`faat`" and "`faaat`" but not "`fat`" or "`faaaat`."

To indicate that a character or expression must:	Anchor	Examples
==========	=====	==========
start a line. (This must occur at the beginning of an expression.)	`^`	"`^hi`" will match "`hi`" only if it is at the beginning of a line.
end a line.	`$`	"`hi$`" will match "`hi`" only if it is at the end of a line.
start a word.	`\<`	"`\<hi`" will match "`hi`" and "`hit`," but not "`phi`."
end a word.	`\>`	"`hi\>`" will match "`hi`" and "`phi`" but not "`hit`."

Brackets may be used to match a range or class of characters. A list of characters enclosed by '`[`' and '`]`' matches any single character in that list; if the first character of the list is the caret '`^`' then it matches any character not in the list. For example, the regular expression "`[0123456789]`" matches any single digit. A range of ASCII characters may be specified by giving the first and last characters, separated by a hyphen. Finally, certain named classes of characters are predefined. Their names are self explanatory, and they are '`[:alnum:]`', '`[:alpha:]`', '`[:cntrl:]`', '`[:digit:]`',

‘[:graph:]‘, ‘[:lower:]‘, ‘[:print:]‘, ‘[:punct:]‘, ‘[:space:]‘, ‘[:upper:]‘, ‘[:word:]‘, ‘[:xdigit:]’. For example, “[[:alnum:]]“ means “[0-9A-Za-z]“ except the latter form is dependent upon the ASCII character encoding, whereas the former is portable, “[[:word:]]“ is equivalent to “\w”, “[[:digit:][:alpha:]_]“ or “[0-9A-Za-z_]“. (Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list.) Most metacharacters lose their special meaning inside lists. To include a literal ‘]’ place it first in the list. Similarly, to include a literal ‘^’ place it anywhere but first. Finally, to include a literal ‘–’ place it last.

A "^" should be used to match a line separator at the beginning of an expression. Otherwise, a "$" should be used to match a line separator. Only the "^" and "$" operators will match a line separator.

Two adjacent (concatenated) regular expressions match a match of the first followed by a match of the second.

Two regular expressions separated by | match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

A regular expression enclosed in parentheses followed by a modifier will match an integer multiple of the enclosed expression. I.e. "(abc)*" will match "abcabc," or "abcabcabc" but will only match the first 6 characters of "abcabcab."

The order of precedence of operators at the same parenthesis level is "[]" then "*+?{}" then concatenation then "|".

An expression will match the longest string it can.

Multiple line matches are limited to five lines.

The reported start and end of a match may be specified by embedding a ‘\s‘ or ‘\e‘ in the regular expression. This will allow using part of the next article in the search pattern. I.e. The search pattern “^$-{30}$$\eFROM:“, will match an article break composed of: a blank line, followed by a line containing 30 dashes, followed by a blank line, followed by a line starting with “FROM:“. Everything following the ‘\e‘ will be included in the following message.

Regular expressions used by DDigest are the same as those used by Yarn with the following exceptions:

1) A "\" may only be followed by one of the following "^.[$()|*+?{\", or "sewW". The sequence "\w" will match any word constituent character, not a "w". A "\" followed by any other character is illegal.

2) A "$" may also be used to match a newline in the middle of a string.

3) The "{}" operator may be used in place of "*+?".

The command line is:

`DDigest` *<config file> <Email file> <rnews file>*

or

`DDigest` *<config file> <Yarn folder> <rnews file>* `-Y`

or

`DDigest` *<config file> <Text file> <rnews file>* `-T`

Where:

<config file>
This file contains the search patterns used to find the digest and to separate it into individual articles.

<Email file>
This file will be found in the soup packet imported into the off-line news reader.

<rnews file>
This file will be generated by DDigest. It could then be imported into the off-line news reader as an rnews file.

<Yarn Folder>
A file contained in Yarns (USER)\mail directory.

<Text file>
A file contained a single digest in text format.

If the *<Email file>*, *<Yarn Folder>*, or *<Text Folder>* name is replaced with a '–', the input will be taken from the standard input. If the *<rnews file>* is replaced with a '–', the output will be sent to the standard output. This will allow the input to be piped from another program and the output to be piped to another program.

To De-Digestify a mailing list digest:

1) Unzip the soup packet containing Email into an empty directory.
2) Decide which file contains Email. This is probably named "`0000000.MSG`" if you are using UQWK to create your SOUP packets. If this does not work, you could find out which one it by looking at the `AREAS` file. Each line contains three fields separated by tabs. The first field is the file name (Without the ".MSG" extension). The second field is the file type. Look for the line containing "`Email`" in the second field. On my system, the line is "`0000000 Email bn`". Therefore the file name is "`0000000.MSG`".

3) Execute the command (substituting your filenames):

```
DDigest list.CFG 0000000.MSG list.MSG
```

4) Import the resultant file:

```
import -r list.MSG
```

Notes:

* Each article extracted from the digest will have unique Message-Id based on the Message-id of the digest. Therefore, if a digest is imported twice, the articles will be recognized as duplicates.

* The program will also insert the following headers into each extracted article based on the digest headers:

  `X-Digest-Subject:` *<Digest subject>*
  `X-Digest-Date:` *<Digest Date>*

* The '`X-Digest-Subject:`', '`X-Digest-Date:`', '`Message-Id:`', and '`Newsgroups:`' lines are added to the beginning of each extracted article.

* The closing will be output as an extra article without a subject.

* The search strings must be contained on one line, but may be any length up to 255 characters long.

* A temporary file is created if either the '`-T`' option is used or if the input is piped into ddigest.

Bob Rush
bobr@mcs.com