demeuk

Release 4.1.0

Netherlands Forensic Institute (NFI)

Aug 17, 2023

CONTENTS

This application is part of the CERBERUS project that has received funding from the European Union's Internal Security Fund - Police under grant agreement No. 82201

Demeuk is a simple tool to clean up corpora (like dictionaries) or any dataset containing plain text strings. Example usecases are: cleaning up language dictionaries, password sets (like for example RockYou) or any file containing plain text strings.

CHAPTER

ONE

TABLE OF CONTENT

1.1 Install

This document describes how to install demeuk.

There are multiple ways to install python packages

- System-wide
- User specific
- Virtual environment

The recommended way to install demeuk is to install it in a virtual environment.

1.1.1 Requirements

- Python 3.6 is required
- Ubuntu 18.04 is the only OS on which demeuk has been tested.

1.1.2 Installing

Virtual environment

```
$ sudo apt install python3-pip
$ sudo pip3 install virtualenv
$ cd <some place where the virtual environment will be created>
$ virtualenv venv-demeuk
$ source venv-demeuk/bin/activate
```

Installing from PyPi

```
$ pip3 install demeuk
```

Installing from source

If for some reason the PyPi is not available, you can build the wheelfile yourself. First create a Virtual environment as described above. *Virtual environment*

```
$ git clone <link to repository>
$ cd demeuk
$ python3 setup.py bdist_wheel
$ pip3 install dist/*.whl
```

Run from source

If for some reason you want to run demeuk from source you only have to install the requirements.

```
$ git clone <link to repository>
$ cd demeuk
$ pip3 install -r requirements.txt
$ python3 bin/demeuk.py --help
```

1.1.3 Upgrading

Upgrading demeuk is quite simple. In case you have installed demeuk through pip and using a virtualenv:

```
$ source venv-demeuk/bin/activate
$ pip3 install demeuk --upgrade
```

In case that you installed demeuk using the source, just rebuild the software and install the wheel file. Pip3 will upgrade the package automatically.

1.2 Usage

This document describes how to usage demeuk.

Please read :*Install*

1.2.1 Basic usage

An example usage for demeuk is to clean up a password list

Download a list, like for example RockYou. The first step you have to document is combine the datafiles into one single file. Using default Linux tooling for this works very well. Next you'll run demeuk on the data to clean it up.

```
$ demeuk.py -i <input file> -o <output file> -l <log file>
-c -j 8 --leak --remove-email
```

So what do all the parameters do? The -i selects the input file. The -o specifies the output file. The -l will specify the log file, by default the log file will only contain information on lines containing invalid characters. For example this can be lines where demeuk was not able to detect the encoding correctly. If you want detailed logging, also include the -v option (verbose logging). The -c specifies that there will be cut based on the first ':' found in a string. The -j indicates that we will be using multithreading and we'll be creating 8 threads. Demeuk has been tested with as many as 48 cores and all cores will be fully used, if IO is not a problem (for example on a fast SSD setup).

The –leak option indicates the enabledment of the following modules: –mojibake, –encode, –newline, –check-controlchar. –mojibake will try to detect and fix encoding issues known as mojibakes. Example of a Mojibake is Sm^rgÂs (Smörgås). This is a very common encoding issue. –encode will enable the encoding detection of demeuk. –newline will remove newlines from lines. –check-controlchar will drop lines containg control-chars.

This set of options was the default for demeuk version 3 and lower.

The –remove-email option will remove simple email addresses from a line. It is useful when a dataset contained line like <something>:<email>:password.

Some datasets contain encoded strings like hex strings (HEX[] format). Those can be decoded using the following example:

```
$ demeuk.py -i <input file> -o <output file> -l <log file> -j all --hex --html
```

The '-j all' option allows demeuk to use all CPU cores in the system. –hex will unhex hex strings. –html will un-htlm htlm escaped passwords.

For additional parsing, demeuk can select based on length of password and even do cutting of the correct field in case of field separated file.

Take for example the entry: testuser:some address:birthday:password

To take the password using demeuk run the following command:

```
$ demeuk.py -i <input file> -o <output file> -c -f4
```

The -c option tells demeuk to cut, and -f4 tell demeuk to select the 4-th field.

Have totally no idea and just what a leak to be fully demeaked? Use the following command:

```
$ demeuk.py -i <input file> -o <output file> -l <log file> -j all --leak-full
```

1.2.2 Standard Options

i input

The input option can be used to select the input file. This can also be a glob pattern. For example: "testdir/*.txt".

o output

The output option can be used to select the output file.

l log

The log option can be used to select to which file a lines needs to be written that are invalid for some reason. There can be multiple reasons, length, encoding and a lot more reason. If the verbose flag is set, this file will also contain any changes, addition or removals that have been made on the line.

j threads

The threads option can be used to speed up the process of demeuking. Of course this option needs to be a number. Do not use more threads then CPU core on your machine. Use the string 'all' to specify to use all cores. Example: -j all

input-encoding

By default demeuk will try to detect the encoding per line. If you already know the input encoding you can specify it using this option. Using this option can speed up the demeuking process significantly. Note: if demeuk fails to decode the line using this encoding, it will still perform the default encoding detection. Thus specifying a not installed encoding will not result in an error.

output-encoding

Probably you do not want to change this option, it defaults to 'en_US.UTF-8'. But in case you want to change the output encoding, use this option. Note, this will change the internal python unicode encoding.

punctuation

Use to set the punctuation that is use by options. For example used by the –remove-punctuation option.

Defaults to all ascci punctuation: ! "#$%&'()*+,-./:;<=>?@[]^_`{|}~

verbose

Use the verbose option to log all the changes made to any line. Note that this will impact the performance of demeuk significantly. Also this will create a large log file.

progress

Use the progress option to enable the progressbar. The progressbar will be displayed for both the chunkify process as well as the demeuking process.

n limit

Limit the number of lines that will be processed. Useful when working with a large dataset and when you want to debug results quickly. Note that the limit parameter is set per thread. This means that if you set the limit to 5 and create 2 threads, 10 lines will be processed. This is not entirely true, if the input file is too small (minimal chunk size) to spawn two threads the limit will only apply to the only thread that could be spawned.

n skip

Skip n lines starting from the start of the file.

1.2.3 Separating options

c cut

Will perform a cut on the line using the delimiter that can be specified. By default it will work with everything AFTER the first delimiter. If the delimiter is present multiple times, the cut will only be performed on the first delimiter. This is in case passwords do contain the delimiter as a character in the password. For example to correctly get the password from the line: <username>:mypassword:is:very:interesting.

f cut-fields

When specifying the –cut command, the cut-fields command can be used to specify which fields needs to be cut. The same syntax as the -f command in the cut binary can be used. This means:

N N'th field, N- from N-th field to end line, N-M, from N-th field to M-th field. -M from start to M-th field.

So examples -f 1-2, will cut field 1 till 2. -f 5 will cut field 5.

cut-before

The cut before option can be used to work with everything before the first delimiter. Basically reverting the default behavior.

d delimiter

Use the delimiter option to cut on a different delimiter. Like cutting on '/'. Default to ':', multiple delimiters can be specified using a ','. If it is needed to split on a comma, make the first delimiter a ','. If you need a comma and multiple delimiters specify the delimiters using ';'. Example: ',;:' would split on ',' and ':'. The order in which they appear matters, the first delimiter will be tested first.

1.2.4 Check modules

check-min-length

Returns only lines that have a specific minimum amount of unicode chars. This is different from the hashcat-utils len.bin, because len.bin works with byte length. The min-length option works with unicode length.

check-max-length

Returns only lines that do not have a specific amount of unicode chars. This is different from the hashcat-utils len.bin, because len.bin works with byte length. The max-length option works with unicode length.

check-case

Check case is a very nifty trick to verify a line is valid printable chars. It will perform a .lower() and .upper() on the line and verify that all characters changed. If some of the char did not change it must mean that there are some punctuation chars inside the line. This option is mostly useful for cleaning up language corpora.

A side effect is that also number will be removed. The check case will ignore some punctuation by default. It will ignore: ” “, “”” and “-“.

check-controlchar

Enable this option to drop lines containg control-chars. Mostly lines containing control-chars are invalid lines, for example lines which are decoded incorrectly.

check-email

Check if a line contains an e-mail address. If so, it drops. It should be noted that this is a every simple regex. Also it is the same regex used for remove-email.

check-hash

Checks if a line is an hash. If so the line is dropped. The regex used are quite simple. One regex check if a line, from start to finish, contains a-f and 0-9’s only. The other checks if the line contains a structure which looks like linux hash. Something like

1fjdfh$qwertyuiopjfsdf

check-mac-address

Checks if a line is a mac address. If so the line is dropped. The line has to be a mac-address from start to finish.

The following line will be dropped:

00:11:22:33:44:55

but a line like:

Dummy:00:11:22:33:44:55

will not be dropped

check-uuid

Checks if a line is an UUID. If this line is a UUID, it will be dropped. The line has to be an UUID from start to finish.

Example

d4662e44-00f1-4ef6-857e-76e3c61604cd

will be dropped

Example

dummy-d4662e44-00f1-4ef6-857e-76e3c61604cd

will not be dropped

check-non-ascii

Checks if a line contains non-ascii chars. It does this by using the 'ascii' encoding builtin Python. If the line does not encode correctly the line is dropped.

check-replacement-character

Checks if a line contains the replacement character. This is the '' Symbol. Mostly when a line contains this char this is an indication that some decoding error happend. The problem is that with this char all information is lost about the original character. So it is very complicated to repair this encoding error. With this option you can drop lines contain this char.

check-starting-with

Checks if a line starts with the argument of check-starting-with. If the line starts with this, the line will be dropped. The string to check can be multiple strings. multiple values are comma-seperated. Example: #,// would skip lines starting with '#' and with '//'.

If you enabled the '–tab' option and you want to drop lines starting with a tab, add ':' to the list of strings to check. '–check starting-with :'. When using –tab tab characters are transfered to ':'.

check-ending-with

Checks if a line ends with the argument of check-ending-with. If the line ends with this, the line will be dropped. The string to check can be multiple strings. multiple values are comma-seperated. Example: #,// would skip lines ending with '#' and with '//'.

If you enabled the '–tab' option and you want to drop lines ending with a tab, add ':' to the list of strings to check. '–check ending-with :'. When using –tab tab characters are transfered to ':'.

check-empty-line

Checks if a line only contains whitespace characters or is empty. If this is true, the line will be dropped.

check-regex

Checks if a line matches a list of regexes. Regexes are comma-seperated. If the line does not matches all of the regexes, the line will be dropped. Example: –check-regex '[a-z],[0-9]' will drop lines that do not atleast contain one lowercase char and one number.

1.2.5 Modify modules

hex

Hashcat convert non-ascii char to hex strings starting with $HEX, but when using corpora for a different attack, the corpora might need to be translated to a different encoding. Thus it is beter to keep one standard and convert HEX strings to plain unicode.

The hex option does this, if a line contains $HEX[], the data between [] will be converted back to a proper byte string and finally be decoded using demeuks decode algorithm.

Small note, if a real passwords contain $HEX[], this will also be converted.

html

Some datasets might contain strings containing html encoded passwords. This can happen because of a implementation of a hash algorithm that encodes passwords submitted by a user in html encoding to support non-ascii characters.

A string like: İSTANBUL will be converted to İSTANBUL. Note, if an password would really contain İ those entries would also be converted. Thus might invalidate some passwords.

This subcommand will only match entries starting with &# followed by alphanumeric and end with a ';'. If you want entries like > to be removed, use the html-named option.

html-named

Html-named option will replace entries like > with '>' and α with the alpha letter. Some of those entries look quite like password entries. Thus use this option with care.

umlaut

In some spellings website the umlaut is not used correct. For example they are encoded as the characters a". This should of course be an a with an umlaut.

non-ascii

Replaces Unicode chars to 7-bit Ascii replacement. For this the following lib is used: https://pypi.org/project/Unidecode/

For example a line like 'kožušček' is replaced to kozuscek.

lowercase

Replace lines like 'Test Test Test' to 'test test test'. Basically lowercasing all words in a line.

title-case

Replace lines like 'test test test' to 'Test Test Test'. Basically uppercasing all words in a line.

mojibake

Use this option to enable trying encoding issues known as mojibakes. Example of a Mojibake is Sm^rgÂs (Smörgås). This is a very common encoding issue. This option will try to detect and fix this issue.

encode

Use this option to enable the encoding guessing of demeuk. This force to decode using the –input-encoding option. Only use this if you are 100% of the input encoding.

tab

If you enable this, demeuk will replace tab characters with ':'. This is useful when cleaning up data from collection leaks. They might contain tab characters and ':' as seperator in the same file.

newline

Enable this option to remove newlines from lines. This can be extra important when using –html or –hex, the decoded lines may contain newline characters. To remove those newline characters, enable this option.

trim

Enable this to let demeuk trim lines. Demuk will removes remove sequences which represent newline characters from beginning and of end of input entry. For example the Ascii sequence 'n' or Html sequence '
'. But in case this sequences are part of a password this option allows to disable this option.

1.2.6 Remove modules

remove-strip-punctuation

Remove starting and trailing punctuation. A line like: test- will be converted to test. This option is useful for language corpora.

remove-punctuation

Remove any punctuation from a line. A line like 'test - hi' will be converted to 'testhi'. What punctuation will be removed can be specified with the '–punctuation' option.

remove-email

The email option will catch lines containing email addresses. like: 12234:test@example.com:password. Not that it is a very simple email filter and many lines will still get through. Especially lines with long subdomains. This option is still very useful for data containing lots of datastructures.

1.2.7 Add modules

add-lower

When working with language dictionaries it can be handy to keep capitalize letters inside your corpora. For example the entry 'Amsterdam' or 'OpenOffice' are likely to be used in this form. But still you probably want 'amsterdam' and 'openoffice' in your corpora. This option keeps both the original format and the lowered part in the corpora.

add-latin-ligatures

In some encoding some characters can be written as one character while they can also be written as two separate chars. Examples of those are ij and ae. This option check if there are any, if there are it will convert the doubled character and add un-double it, but keeping the original in the corpora as well.

So in case: cijfer is present, both cijfer and cijfer will be added.

add-umlaut

In some spellings website the umlaut is not used correct. For example the characters a" are in those sites. This should of course be an a with an umlaut.

add-split

In some language dictionaries some words are coupled that might be interesting to also add uncoupled.

Example: 3D-printer, add split will split the word and add: 3D, printer and 3D-printer to the corpora. Note: Add-split will not perform a length check that was specified using the –min-length option. It only checks if the length of a split part is longer then 1 unicode character.

add-without-punctuation

If a line contains punctuations, a variant will be added without the punctuations. Example a line like: 'test-123' will be kept, plus 'test123' will be added. Which punctuation will be removed can be specified with the –punctuation option.

1.2.8 Macro modules

g googlengram

In case you are working with the googlengram's, this option is a macro for:

- Don't remove control characters or tabs
- Don't detect mojibakes
- Do detect encoding
- Strip ngram tagging

When using –googlengram, don't using any other options.

Basically it will strip the tags like: _NOUN_ or _ADJ

leak

The leak option will enable the following modules:

- mojibake
- encode
- newline
- check-controlchar

leak-full

The leak-full option will enable the following modules:

- mojibake
- encode
- newline
- check-controlchar
- hex
- html
- html-named
- check-email
- check-hash
- check-mac-address
- check-uuid
- check-replacement-character

- check-empty-line

1.3 Design

This document will describe how the internal of demeuk are designed. It gives some insight on how the application works. Mostly it is useful in case you are working with a bug or don't understand why something is happening and it is a must read for anyone adding features to demeuk.

1.3.1 Threading

To start of, the input file is counted by the main processes. It will split the input files in chunks. It does so by reading the file per 1 KB. After reading 1 KB it will search for the next newline after the 1 KB. It will check the file pointer byte offset. It will then read again 1 KB and search for the first new line after that. This starting and ending offsets are stored in a list and threads will read useful the list to determine what to work on.

The size of 1 KB is used to reduce memory load and was found to be a solid number for good performance.

Next a thread will open the input file and seek to the start offset. It will read the remaining byte to the end offset and starts processing the lines.

Will processing the input file, the thread will create a temp file inside the folder 'demeuk_tmp' inside the current working directory. Inside this temp file intermediate results will be written to reduce memory usages. Note: many thread will cause a significant IO storm. If you see a lot of IO wait, reduce the amount of threads or replace you disks with faster disks.

Once all threads are done, the main thread will combine all of the results in the temp folder. You should note that the order inside the final output will be completely un ordered and thus if you want to have a sorted list you need to sort it yourself.

1.3.2 Encoding detection

So, a thread has opened a file, it will start reading it using the splitlines() python function. This means the line will be splitted on: line feed, carriage return, LF + CR, formfeeds, file separator, etc. See https://docs.python.org/3/library/stdtypes.html for more information.

Next, when '–tab' is enabled all tabs will be converted to ':' greedy. This is to have a single cut/splitting char. This is done on binary level.

Next, we arrive at one of the most important things of this application. The encoding detecting enable this with '–encode'. Some dataset are a combination of different sources. This means EVERY line can have a different encoding. People or applications tend to make a lot of errors in encoding, as does this application. Demeuk tries its best to detect and correct as much as possible, but there will for sure be some weird case where it fails to do so. By default the application will try to decode the data using UTF-8.

So we start by checking if we have a default encoding to try. This is either UTF-8 or supplied by the user. If the line decodes and there does not appear to be control character inside the line we can assume that the detection went correctly. Also, if you supply a list of input encodings. First put multibyte encodings first. Because single byte encodings will cause false positives.

If that fails we run the detect function of the chardet library. Note: first the cchardet library was implemented, but this library resulted in too many wrongly encoded lines. Inside the tests of demeuk there are lot of edge cases which were found and corrected. So if you change something in the encoding detection please run the tests to verify that you have not broken something.

If it managed detect any encoding, it will try to decode this line. If no unicode error happens we assume that we got some result.

Next we try to fix mojibakes, for this enable the –mojibake option basically, we might have decoded the string incorrectly and now correct some of the common errors. For this we use the FTFY library.

1.3.3 Modules

After a line has been decoded correctly demeuk will start to run all the modules. Demeuk consist of 4 different type of modules.

- Clean modules. Those modules modify something in a line. For example replace tab character with ':'. The commandline parameters will have the name of the module without a prefix.
- Add modules. Those modules will modify something in a line, but keep the original line aswell. For example, add a lower case variant of a line. These modules will have the commandline parameters start with 'add-' prefix.
- Check modules. Those modules will check if a line passes some test. For example a minimal length check. The commandline parameters start with the 'check-' prefix. If a line fails the check, the line is dropped.
- Remove modules. Those modules will remove specific parts of a line and does this in place. For example punctuation needs to be removed, those modules will be used. The commandline parameters will start with the 'remove-' prefix.

The name that a module has on the commandline will mean that the function inside the source code must also has the exact same name. Only clean module will start with the 'clean_' prefix to prevent name clashes with default functions.

Note that when any add option is used, any other modules (like clean, check, remove AND even add) will be ran on the modified line again. This might result in creating an loop if it keeps creating new lines. So be careful with using those options.

For now there is no specific order in which the module type will run. Apart from the add modules, which will always run last. If someone find a specific use case for which the order needs to be configured; please submit a bug.

Another note on the add modules and threading. Lines are dedicated to different threads based on a configured chunk size. When additional lines are added, all other modules will run again on the line. The thread that created the new line will also run those modules again. Meaning that if one thread creates a lot of different new lines that thread might be busier then other threads. But because the chunksize is quite small, this will probably not be an issue. If this is an issue for someone please submit a bug.

1.4 API Reference

This chapter is for developers of demeuk, it contains the API functions.

1.5 Demeuk-api

```
Demeuk - a simple tool to clean up corpora

Usage:
    demeuk [options]

Examples:
    demeuk -i inputfile.tmp -o outputfile.dict -l logfile.txt
```

(continues on next page)

(continued from previous page)

```
    demeuk -i "inputfile*.txt" -o outputfile.dict -l logfile.txt
    demeuk -i "inputdir/*" -o outputfile.dict -l logfile.txt
    demeuk -i inputfile -o outputfile -j 24
    demeuk -i inputfile -o outputfile -c -e
    demeuk -i inputfile -o outputfile --threads all

Standard Options:
    -i --input <path to file>       Specify the input file to be cleaned, or provide a␣
↪glob pattern
    -o --output <path to file>      Specify the output file name.
    -l --log <path to file>         Optional, specify where the log file needs to be␣
↪writen to
    -j --threads <threads>          Optional, demeuk doesn't use threads by default.␣
↪Specify amount of threads to
                                    spawn. Specify the string 'all' to make demeuk auto␣
↪detect the amount of threads
                                    to start based on the CPU's.
                                    Note: threading will cost some setup time. Only␣
↪speeds up for larger files.
    --input-encoding <encoding>     Forces demeuk to decode the input using this␣
↪encoding (default: en_US.UTF-8).
    --output-encoding <encoding>    Forces demeuk to encoding the output using this␣
↪encoding (default: en_US.UTF-8).
    -v --verbose                    When set, the logfile will not only contain lines␣
↪which caused an error, but
                                    also line which were modified.
    --progress                      Prints out the progress of the demeuk process.
    -n --limit <int>                Limit the number of lines per thread.
    -s --skip <int>                 Skip <int> amount of lines per thread.
    --punctuation <punctuation>     Use to set the punctuation that is use by options.␣
↪Defaults to:
                                    ! "#$%&'()*+,-./:;<=>?@[\]^_`{|}~
    --version                       Prints the version of demeuk.

Separating Options:
    -c --cut                        Specify if demeuk should split (default splits on ':
↪'). Returns everything
                                    after the delimiter.
    --cut-before                    Specify if demeuk should return the string before␣
↪the delimiter.
                                    When cutting, demeuk by default returns the string␣
↪after the delimiter.
    -f --cut-fields <field>         Specifies the field to be returned, this is in the
↪'cut' language thus:
                                    N N'th field, N- from N-th field to end line, N-M,␣
↪from N-th field to M-th
                                    field. -M from start to M-th field.
    -d --delimiter <delimiter>      Specify which delimiter will be used for cutting.␣
↪Multiple delimiters can be
                                    specified using ','. If the ',' is required for␣
↪cutting, escape it with a
                                    backslash. Only one delimiter can be used per line.
```

(continues on next page)

(continued from previous page)

```
Check modules (check if a line matches a specific condition):
    --check-min-length <length>     Requires that entries have a minimal requirement of
↪<length> unicode chars
    --check-max-length <length>     Requires that entries have a maximal requirement of
↪<length> unicode chars
    --check-case                    Drop lines where the uppercase line is not equal to␣
↪the lowercase line
    --check-controlchar             Drop lines containing control chars.
    --check-email                   Drop lines containing e-mail addresses.
    --check-hash                    Drop lines which are hashes.
    --check-mac-address             Drop lines which are MAC-addresses.
    --check-uuid                    Drop lines which are UUID.
    --check-non-ascii               If a line contain a non ascii char e.g. ü or ç (or␣
↪everything outside ascii
                                    range) the line is dropped.
    --check-replacement-character   Drop lines containing replacement characters ''.
    --check-starting-with <string>  Drop lines starting with string, can be multiple␣
↪strings. Specify multiple
                                    with as comma-seperated list.
    --check-ending-with <string>    Drop lines ending with string, can be multiple␣
↪strings. Specify multiple
                                    with as comma-seperated list.
    --check-empty-line              Drop lines that are empty or only contain whitespace␣
↪characters
    --check-regex <string>          Drop lines that do not match the regex. Regex is a␣
↪comma seperated list of
                                    regexes. Example: [a-z]{1,8},[0-9]{1,8}

Modify modules (modify a line in place):
    --hex                           Replace lines like: $HEX[41424344] with ABCD.
    --html                          Replace lines like: &#351;ifreyok with şifreyok.
    --html-named                    Replace lines like: &#alpha; Those structures are␣
↪more like passwords, so
                                    be careful to enable this option.
    --lowercase                     Replace line like 'This Test String' to 'this test␣
↪string'
    --title-case                    Replace line like 'this test string' to 'This Test␣
↪String'
    --umlaut                        Replace lines like ko"ffie with an o with an umlaut.
    --mojibake                      Fixes mojibakes, which means lines like Sm^rgÂs will␣
↪be fixed to Smörgås.
    --encode                        Enables guessing of encoding, based on chardet and␣
↪custom implementation.
    --tab                           Enables replacing tab char with ':', sometimes leaks␣
↪contain both ':' and '\t'.
    --newline                       Enables removing newline characters (\r\n) from end␣
↪and beginning of lines.
    --non-ascii                     Replace non ascii char with their replacement␣
↪letters. For example ü
                                    becomes u, ç becomes c.
    --trim                          Enables removing newlines representations from end␣
```

(continues on next page)

(continued from previous page)

```
↪and beginning. Newline
                                representations detected are '\\n', '\\r', '\n', '\r
↪', '<br>', and '<br />'.

Add modules (Modify a line, but keep the original as well):
    --add-lower                 If a line contains a capital letter this will add␣
↪the lower case variant
    --add-latin-ligatures       If a line contains a single ligatures of a latin␣
↪letter (such as ij), the line
                                is correct but the original line contain the␣
↪ligatures is also added to output.
    --add-split                 split on known chars like - and . and add those to␣
↪the final dictionary.
    --add-umlaut                In some spelling dicts, umlaut are sometimes written␣
↪as: o" or i" and not as
                                one char.
    --add-without-punctuation   If a line contains punctuations, a variant will be␣
↪added without the
                                punctuations

Remove modules (remove specific parts of a line):
    --remove-strip-punctuation  Remove starting and trailing punctuation
    --remove-punctuation        Remove all punctuation in a line
    --remove-email              Enable email filter, this will catch strings like
                                1238661:test@example.com:password
Macro modules:
    -g --googlengram            When set, demeuk will strip universal pos tags: like␣
↪_NOUN_ or _ADJ
    --leak                      When set, demeuk will run the following modules:
                                    mojibake, encode, newline, check-controlchar
                                This is recommended when working with leaks and was␣
↪the default bevarior in
                                demeuk version 3.11.0 and below.
    --leak-full                 When set, demeuk will run the following modules:
                                    mojibake, encode, newline, check-controlchar,
                                    hex, html, html-named,
                                    check-hash, check-mac-address, check-uuid, check-
↪email,
                                    check-replacement-character, check-empty-line
```

bin.demeuk.**add_latin_ligatures**(*line*)

Returns the line cleaned of latin ligatures if there are any.

Param:

line (unicode)

Returns

False if there are not any latin ligatures Corrected line

bin.demeuk.**add_lower**(*line*)

Returns if the upper case string is different from the lower case line

Param:

line (unicode)

Returns
False if they are the same Lowered string if they are not

bin.demeuk.**add_split**(*line*, *punctuation=(' ', '-', '\\.')*)

Split the line on the punctuation and return elements longer then 1 char.

Param:
line (unicode)

Returns
split line

bin.demeuk.**add_without_punctuation**(*line*, *punctuation*)

Returns the line cleaned of punctuation.

Param:
line (unicode)

Returns
False if there are not any punctuation Corrected line

bin.demeuk.**check_case**(*line*, *ignored_chars=(' ', '"', '-')*)

Checks if an uppercase line is equal to a lowercase line.

Param:
line (unicode) ignored_chars list(string)

Returns
true if uppercase line is equal to uppercase line

bin.demeuk.**check_character**(*line*, *character*)

Checks if a line contains a specific character

Params:
line (unicode)

Returns
true if line does contain the specific character

bin.demeuk.**check_controlchar**(*line*)

Detects control chars, returns True when detected

Params:
line (Unicode)

Returns
Status, String

bin.demeuk.**check_email**(*line*)

Check if lines contain e-mail addresses with a simple regex

Params:
line (unicode)

Returns
true is line does not contain email

bin.demeuk.**check_empty_line**(*line*)

Checks if a line is empty or only contains whitespace chars

Params:
line (unicode)

Returns
true of line is empty or only contains whitespace chars

bin.demeuk.**check_ending_with**(*line*, *strings*)

Checks if a line ends with specific strings

Params:
line (unicode) strings[str]

Returns
true if line does end with one of the strings

bin.demeuk.**check_hash**(*line*)

Check if a line contains a hash

Params:
line (unicode)

Returns
true if line does not contain hash

bin.demeuk.**check_length**(*line*, *min=0*, *max=0*)

Does a length check on the line

Params:
line (unicode) min (int) max (int)

Returns
true if length is ok

bin.demeuk.**check_mac_address**(*line*)

Check if a line contains a MAC-address

Params:
line (unicode)

Returns
true if line does not contain a MAC-address

bin.demeuk.**check_non_ascii**(*line*)

Checks if a line contains a non ascii chars

Params:
line (unicode)

Returns
true if line does not contain non ascii chars

bin.demeuk.**check_regex**(*line*, *regex*)

Checks if a line matches a list of regexes

Params:
line (unicode) regex (list)

Returns
true if all regexes match false if line does not match regex

bin.demeuk.**check_starting_with**(*line*, *strings*)

Checks if a line start with a specific strings

Params:
line (unicode) strings[str]

Returns
true if line does start with one of the strings

bin.demeuk.**check_uuid**(*line*)

Check if a line contains a UUID

Params:
line (unicode)

Returns
true if line does not contain a UUID

bin.demeuk.**chunkify**(*fname*, *config*, *size=1048576*)

bin.demeuk.**clean_add_umlaut**(*line*)

Returns the line cleaned of incorrect umlauting

Param:
line (unicode)

Returns
Corrected line

bin.demeuk.**clean_cut**(*line*, *delimiters*, *fields*)

Finds the first delimiter and returns the remaining string either after or before the delimiter.

Params:
line (unicode) delimiters list(unicode) fields (unicode)

Returns
line (unicode)

bin.demeuk.**clean_encode**(*line*, *input_encoding*)

Detects and tries encoding

Params:
line (bytes)

Returns
Decoded UTF-8 string

bin.demeuk.**clean_googlengram**(*line*)

Removes speechtags from line specific to the googlengram module

Param:
line (unicode)

Returns
line (unicode)

bin.demeuk.**clean_hex**(*line*)

Converts strings like '$HEX[]' to proper binary

Params:
line (bytes)

Returns
line (bytes)

bin.demeuk.**clean_html**(*line*)

Detects html encode chars and decodes them

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_html_named**(*line*)

Detects named html encode chars and decodes them

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_lowercase**(*line*)

Replace all capitals to lowercase

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_mojibake**(*line*)

Detects mojibake and tries to correct it. Mojibake are string that are decoded incorrectly and then encoded incorrectly. This results in strings like: Ãºnico which should be único.

Param:
line (str)

Returns
Cleaned string

bin.demeuk.**clean_newline**(*line*)

Delete newline characters at start and end of line

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_non_ascii**(*line*)

Replace non ascii chars with there ascii representation.

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_tab**(*line*)

Replace tab character with ':' greedy

Params:
line (bytes)

Returns
line (bytes)

bin.demeuk.**clean_title_case**(*line*)

Replace words to title word (uppercasing first letter)

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_trim**(*line*)

Delete leading and trailing character sequences representing a newline from beginning end end of line.

Params:
line (Unicode)

Returns
line (Unicode)

bin.demeuk.**clean_up**(*filename*, *chunk_start*, *chunk_size*, *config*)

Main clean loop, this calls all the other clean functions.

Parameters
line (*bytes*) – Line to be cleaned up

Returns
(str(Decoded line), str(Failed line))

bin.demeuk.**main**()

bin.demeuk.**remove_email**(*line*)

Removes e-mail addresses from a line.

Params:
line (unicode)

Returns
line (unicode)

bin.demeuk.**remove_punctuation**(*line*, *punctuation*)

Returns the line without punctuation

Param:
line (unicode) punctuation (unicode)

Returns
line without start and end punctuation

bin.demeuk.**remove_strip_punctuation**(*line*, *punctuation*)

Returns the line without start and end punctuation

Param:
line (unicode)

Returns
line without start and end punctuation

bin.demeuk.**try_encoding**(*line*, *encoding*)

Tries to decode a line using supplied encoding

Params:
line (Byte): byte variable that will be decoded encoding (string): the encoding to be tried

Returns
False if decoding failed String if decoding worked

PYTHON MODULE INDEX

INDEX