# unitils Documentation

***Release 0.1.0***

**iLoveTux**

September 27, 2016

# Contents

CHAPTER 1

# Introduction

## 1.1 What is it and why do I care?

Unitils has been incredibly useful for my co-workers and myself. They are simplified, altered forms of common and useful utilities you are likely to find on most Unix-like operating systems. They are written as command line utilities, but also present a single Python generator which can be used from within Python without "shelling" the command out.

Because of the simplified nature of these utilities along with the compromises we have made. We wished to differentiate ourselves from similar commands which were our inspiration. Each of our utilities appends ".py" to the end of the command. For instance, our version of *grep* can be invoked with the *grep.py* command.

For instance, *grep.py* is designed to be used just like this:

```
$ grep.py -i 'warn' /var/log/*.log
```

And from Python it can be used like this:

```
from glob import iglob
from unitils import grep

for match in grep("warn", iglob('/var/log/*.log'), ignore_case=True):
    print(match)
```

## 1.2 Why should I use it?

Unitils is a collection of useful utilities which have been re-written to be simple and to provide a CLI as well as a Python API.

Unitils was written to be:

- Fast, everything is an generator (where possible) and strives to be as efficient in both memory and cpu time.
- Tested, Unittests are important and we strive for high test coverage.
- Cross Platform, Written in Python these utilities can run on Windows, Linux and Mac OSx.
- Provides an API to use these utilities in Python, cross-platform and without "shelling out".
- Open Source, This project is released under the GPLv3

## 1.3 How does it work?

Each command we target, we create a Python generator which yields the output and send it to stdout. So we in effect have native, memory efficient access to many common utilities directly from within Python. We then wrap a command line interface around this generator tp provide our users with a convenient cross-platform utility.

## 1.4 How do I get it?

To get the most supported version:

```
$ pip install unitils
```

To get the latest version:

```
$ pip install https://github.com/ilovetux/unitils/archive/master.zip
```

For the nightlies:

```
$ pip install https://github.com/ilovetux/unitils/archive/dev.zip
```

## 1.5 How do I run the tests?

You can clone the repository and use the following command:

```
$ make test
```

or alternately:

```
$ python setup.py nosetests
```

In general, the master branch is what is available on PyPI.

## 1.6 What is this compatible with?

Unitils is tested and confirmed to work with

- Python 3.5
- Python 3.4
- Python 3.3
- Python 2.7
- pypy

Unitils should work on all platforms on which Python runs.

## 1.7 What is the current list of utilities provided by unitils?

- cat
- cp

- find
- grep
- head
- ls
- mv
- watch
- wc
- which

## 1.8 What is on the list to be done?

See this issue for the state of our current prgress.

## 1.9 How can I help?

You can do all the github type things, submit an issue in our issue tracker or fork and submit a pull request. If none of that appeals to you, you can always send me an email personally at me@ilovetux.com

### 1.9.1 Introduction

#### What is it and why do I care?

Unitils has been incredibly useful for my co-workers and myself. They are simplified, altered forms of common and useful utilities you are likely to find on most Unix-like operating systems. They are written as command line utilities, but also present a single Python generator which can be used from within Python without “shelling” the command out.

Because of the simplified nature of these utilities along with the compromises we have made. We wished to differentiate ourselves from similar commands which were our inspiration. Each of our utilities appends ”.py” to the end of the command. For instance, our version of *grep* can be invoked with the *grep.py* command.

For instance, *grep.py* is designed to be used just like this:

```
$ grep.py -i 'warn' /var/log/*.log
```

And from Python it can be used like this:

```
from glob import iglob
from unitils import grep

for match in grep("warn", iglob('/var/log/*.log'), ignore_case=True):
    print(match)
```

#### Why should I use it?

Unitils is a collection of useful utilities which have been re-written to be simple and to provide a CLI as well as a Python API.

Unitils was written to be:

- Fast, everything is an generator (where possible) and strives to be as efficient in both memory and cpu time.
- Tested, Unittests are important and we strive for high test coverage.
- Cross Platform, Written in Python these utilities can run on Windows, Linux and Mac OSx.
- Provides an API to use these utilities in Python, cross-platform and without "shelling out".
- Open Source, This project is released under the GPLv3

## How does it work?

Each command we target, we create a Python generator which yields the output and send it to stdout. So we in effect have native, memory efficient access to many common utilities directly from within Python. We then wrap a command line interface around this generator tp provide our users with a convenient cross-platform utility.

## How do I get it?

To get the most supported version:

```
$ pip install unitils
```

To get the latest version:

```
$ pip install https://github.com/ilovetux/unitils/archive/master.zip
```

For the nightlies:

```
$ pip install https://github.com/ilovetux/unitils/archive/dev.zip
```

## How do I run the tests?

You can clone the repository and use the following command:

```
$ make test
```

or alternately:

```
$ python setup.py nosetests
```

In general, the master branch is what is available on PyPI.

## What is this compatible with?

Unitils is tested and confirmed to work with

- Python 3.5
- Python 3.4
- Python 3.3
- Python 2.7
- pypy

Unitils should work on all platforms on which Python runs.

#### What is the current list of utilities provided by unitils?

- cat
- cp
- find
- grep
- head
- ls
- mv
- watch
- wc
- which

#### What is on the list to be done?

See this issue for the state of our current prgress.

#### How can I help?

You can do all the github type things, submit an issue in our issue tracker or fork and submit a pull request. If none of that appeals to you, you can always send me an email personally at me@ilovetux.com

### 1.9.2 Rationale

#### Abstract

This document outlines in detail why I believe this library is a worthwhile investment in time and resources.

##### Where the idea originated

There is a slight disconnect between DevOps engineers (particularly Operations personele, Administration personele and Security Analysists) caused by a difference in language, culture and tools. Over and over, people are being asked to transition to a DevOps work cycle but something as simple as learning the new tools and mindset can be overwhelming. Most non-development IT professionals are familiar with basic linux commands, so we decided to explore that.

A lot of the work being done in the DevOps space are written in Python, and because Python is a great language for automation, we decided that it would be almost a gift to allow people making the DevOps transition. Automation is nothing new and lots of people have experience making shell scripts or batch files to acomplish the same task. If these people were given a familiar toolset with which to work in Python their transition will be just that much easier.

At the same time, we realized that "shelling out" commands was a real option, but the overhead of spawing shells and the limitation of using OS-specific system calls makes this unusable for cross-platform automation. Instead what we needed was a native Python function to mirror how these commands work on the outside.

About the time we made this realization, a few of my co-workers along with myself found ourselves working in a Windows environment. We agonized about having our basic toolset taken away from us (Imagine a carpenter working in a shop that forbids hammers). We would have loved to just be able to tail a file and grep the logs (I know there are Windows ways of doing the above tasks, but I only make use of them every couple of years).

This is when our idea was made to not only make a library which provides functions mirroring Linux commands, but to also provide these functions with a Command Line Interface. The dual interfaces (programatic and command line) allow great flexibility.

#### The commands we are targeting

Please see this issue for the current state of our progress.

## 1.9.3 CLI

### cat.py(1)

#### Name

cat.py - Concatenate files and print on standard output.

#### SYNOPSIS

```
cat.py [OPTIONS]... [FILE]...
```

#### DESCRIPTION

Concatenate FILE(s) to standard output.

When no FILE or when FILE is “-”, read standard input.

```
-h, --help
    Print usage message and exit
--version
    output version information and exit
-n, --number
    Number all output lines.
```

#### EXAMPLES

#### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with cat as released by the Free Software Foundation.

#### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software.

#### COPYRIGHT

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

### cp.py(1)

#### Name

cp.py - Copy a file or directory

#### SYNOPSIS

```
cp.py [OPTIONS] SRC DST
```

#### DESCRIPTION

cp.py copies a source file or directory to a destination. If dst is a directory, then a file with the same name as the original will be placed within the directory. If src is a directory, you should specify –recursive as this will allow the directory to be copied in full.

This is a simplified clone of the well-known cp command. Not all options for the original will be available for this command.

#### OPTIONS

**Generic Program Information**

```
--help
    Output a usage message and exits.
-R, --recursive
    Recursively copy the contents of src to dst
-n, --no-clobber
    If specified, files will not be copied if they already exist at dst
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with cp as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

### SEE ALSO

None yet

### NOTES

None yet

## find.py(1)

### Name

find.py - Search for files in a directory heirarchy

### SYNOPSIS

```
find.py [OPTIONS...] [PATH]
```

### DESCRIPTION

find.py currently differs greatly from the GNU implementation of find. We do not currently evaluate expressions. We believe that we can cover most common use cases with options rather than with a Domain Specific Language (DSL), if this is found not to be the case, we will re-evaluate.

We try as much as possible to make the usage feel like the usage of the GNU find utility.

### OPTIONS

```
-iname=INAME
    The case-insensitive name spec (glob pattern) to search for.
-name=NAME
    The case-sensitive name spec (glob pattern) to search for.
-type=TYPE
    The type of file to look for. TYPE can be any of the following:

    b   block (buffered) special
    c   character (unbuffered) special
    d   directory
    p   named pipe (FIFO)
    f   regular file
    l   symbolic link
    s   socket
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with find as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

### grep.py(1)

#### Name

grep.py - print lines matching a pattern

#### SYNOPSIS

```
grep.py [OPTIONS] PATTERN [FILE...]
```

#### DESCRIPTION

grep.py searches the named input FILEs for lines containing a match to the given PATTERN. If no files are specified, or if the file "-" is given, grep.py searches standard input. By default, grep.py prints the matching lines.

This is a simplified clone of the well-known grep command. Not all options for the original will be available for this command. In addition, regular expressions are handled differently by grep.py than by grep. Particularly only one flavor of regular expressions are supported, this is because Python's regular expression is used for ease and speed of development.

#### OPTIONS

##### Generic Program Information

```
--help
    Output a usage message and exits.
-V, --version
    Output the version number of grep.py and exit
```

##### Matching Control

```
-i, --ignore-case
    Ignore case distinctions in both the PATTERN and the input files.
-v, --invert-match
    Invert the sense of matching, to select non-matching lines
```

#### General Output Control

```
--color[=WHEN]
    WHEN can be "never", "always" or "auto".  On *nix systems, ANSI escape
    sequences are used to achieve terminal coloring. On Windows systems,
    these ANSI escape sequences are intercepted and the appropriate system
    API calls are made to color the text. No environment variables are
    interpreted to control the color.
```

#### Output Line Prefix Control

```
-H, --with-filename
    Print the filename for each match.
-n, --line-number
    Prefix each line of output with a 1-based line number within
    its input file.
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with grep as released by the Free Software Foundation with the exception of the multiple regular expression engines. We will stick to Python's native regular expression engine as we believe that writing regular expression engines is beyond the scopeof this project.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

### SEE ALSO

None yet

### NOTES

None yet

## head.py(1)

### Name

head.py - Output the first part of files

### SYNOPSIS

```
head.py [OPTIONS]... [FILE]...
```

### DESCRIPTION

head.py will send the first 10 lines of each file to stdout. If more than one file is specified each will be preceded by a header containing the filename

This is a simplified clone of the well-known head command. Not all options for the original will be available for this command.

### OPTIONS

#### Generic Program Information

```
--help
    Output a usage message and exits.
-n, --lines
    The number of lines to print
-q, --quiet
    Never print the header with the filename
-v, --verbose
    Always print the header with the filename
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with head as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

#### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

#### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

### ls.py(1)

#### Name

ls.py - List directory contents

#### SYNOPSIS

```
ls.py [OPTIONS...] [FILE...]
```

#### DESCRIPTION

List information about the FILEs (the current directory by default). Sort entries alphabetically.

This is currently extremely limited compared to the GNU ls command, but improvements are expected to be done soon. If you need a specific feature to be completed please don't hesitate to open an issue in our issue tracker.

#### OPTIONS

```
-a, --all
    Do not ignore entries starting with .
-A, --almost-all
    do not list implied . and ..
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with ls as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

### SEE ALSO

None yet

### NOTES

None yet

## mv.py(1)

### Name

> mv.py - Recursively move src to dst

### SYNOPSIS

```
mv.py [OPTIONS] SRC DST
```

### DESCRIPTION

Recursively move src to dst.

This is currently extremely limited compared to the GNU ls command, but improvements are expected to be done soon. If you need a specific feature to be completed please don't hesitate to open an issue in our issue tracker.

### OPTIONS

```
-h, --help
    Prints a usage message and exits
```

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with mv as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

### SEE ALSO

None yet

### NOTES

None yet

## watch.py(1)

### Name

watch.py - execute a program periodically, displaying the results

### SYNOPSIS

```
wc.py [OPTIONS...] COMMAND
```

### DESCRIPTION

watch.py runs COMMAND repeatedly, displaying its output and errors. This allows you to watch the program output change over time. By default, the program is run every two seconds. By default, watch.py will run until interrupted.

NOTE: This program differs significantly in execution from the GNU version of watch. This is because I have not been able to find a way to go fullscreen within a Windows command prompt. Instead, we clear the screen before displaying the new output (cls on Windows and clear on *nix). If you know of a way to achieve the desired results in a cross-platform way within Python, please open an issue in our issue tracker or better yet, open a pull request

### EXAMPLES

### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with watch as released by the Free Software Foundation.

### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

### wc.py(1)

#### Name

wc.py - Print newline, word and byte counts for each file

#### SYNOPSIS

```
wc.py [OPTIONS...] [FILES...]
```

#### DESCRIPTION

Print newline, word and byte counts for each FILE, and a total line if more than one file is specified. A word is a non-zero-length sequence of characters delimited by whitespace.

With no FILE or when FILE is “-”, read standard input.

The options below may be used to select which counts are printed, always in the following order: newline, word, character, byte, maximum line length.

```
-c, --bytes
    print the byte counts

-m, --chars
    print the character counts

-l, --lines
    print the newline counts

-L, --max-line-length
    print the maximum display width

-w, --words
    print the word counts

-h, --help display this help and exit

--version
      output version information and exit
```

#### EXAMPLES

#### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with wc as released by the Free Software Foundation.

#### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

#### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

#### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

### which.py(1)

#### Name

which.py - Search PATH for cmd and return the first instance

#### SYNOPSIS

```
which.py [OPTIONS] CMD
```

#### DESCRIPTION

Searches the PATH for cmd and prints the first occurance. This means that the path printed will be the command invoked when cmd is issued.

This is currently extremely limited compared to the GNU ls command, but improvements are expected to be done soon. If you need a specific feature to be completed please don't hesitate to open an issue in our issue tracker.

#### OPTIONS

```
-h, --help
    Prints a usage message and exits
-a, --all
    Prints all matches, not just the first one
```

#### EXAMPLES

#### OVERVIEW

This is a work-in-progress, we aim to become feature-compatible with which as released by the Free Software Foundation.

#### ENVIRONMENT VARIABLES

No environment variables affect the behavior of this software

#### EXIT STATUS

We aim to take advantage of exit status, but currently do not. The exit status will be 0 unless an error occurs in which case it will be non-zero.

#### COPYRIGHT

Copyright 2016 Clifford Bressette IV (iLoveTux).

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

#### BUGS

Bugs can be reported in our issue tracker. This is also the correct place for feature requests.

#### SEE ALSO

None yet

#### NOTES

None yet

Please see this issue for the current state of our progress.

#### Overview

The CLI utilites provided by unitils are designed to mirror their GNU equivalents whenever possible. Sometimes we will make concessions due to time-to-market concerns, efficiency concerns or simply thinking that YAGNI.

Below follows a listing of commands along with the options which control how they work along with descriptions.

#### Stay Tuned

### 1.9.4 API

`unitils.`**`cat`**(*files*, *number=False*)
iterate through each file in files and yield each line in turn.

**Parameters**

- **`files`** (*list of open file-like objects*) – The files to concatenate
- **`number`** (*boolean*) – If True, yield two-tuples of (line_number, line)

`unitils.`**`cp`**(*src*, *dst*, *no_clobber=False*, *recursive=False*)
Copy src to dst.

**Parameters**

- **`src`** (*str*) – The file(s) to copy
- **`dst`** (*str*) – The destination for src
- **`no_clobber`** (*boolean*) – If True, do not overwrite files if they already exist
- **`recursive`** (*boolean*) – If True recursively copy the contents of src to dst

`unitils.`**`find`**(*path='.'*, *name=None*, *iname=None*, *ftype='*'*)
Search for files in a directory heirarchy.

This is dramatically different from the GNU version of find. There is no Domain Specific language.

**Parameters**

- **`path`** (*str*) – The directory to start in
- **`name`** (*str*) – The name spec (glob pattern) to search for
- **`iname`** (*str*) – The case-insensitive name spec (glob pattern) to search for
- **`ftype`** (*str*) – The type of file to search for must be one of b, c, d, p, f, k, s or *

`unitils.`**`grep`**(*expr*, *files*, *line_numbers=False*, *filenames=False*, *color=False*, *invert_match=False*, *ignore_case=False*)
search the contents of files for expr, yield the results

files can be a filename as str, a list of filenames, a file-like object or a list of file-like objects. In any case, all files will be searched line-by-line for any lines which contain expr which will be yielded.

This does not support sending text in directly to search, the reason is that this operation is fairly simple in Python:

```
import re

expr = re.compile(r"^\d+\s\w+")
matches = (l for l in text.splitlines() if expr.search(line))
for line in matching_lines:
    print(line)
```

**Parameters**

- **files** (*str, list, file*) – files to search for expr
- **expr** (*str, compiled regex*) – the regular expression to search for
- **line_numbers** (*bool*) – If True, line numbers will be prepended to results
- **filenames** (*bool*) – If True, filenames will be prepended to results

unitils.**head**(*files*, *lines=10*, *verbose=False*, *quiet=False*)

Read the first 10 lines (by default) of each file in files.

As this is supposed to imitate the behavior of head, but also be used as a Python callable, some liberties have been taken to accomodate the functionality.

If one file is passed in, a generator is returned which will yield the first n lines in the file.

If more than one file is passed in, a dict is returned keyed by filename mapping to a generator which will yield the first n lines of that file.

**Parameters**

- **files** (*str list*) – The files to examine
- **lines** (*int*) – The number of lines to yield from each file
- **verbose** (*boolean*) – Always include the filename (as described above)
- **quiet** (*boolean*) – Never include the filename (as described above)

unitils.**ls**(*path='.'*, *_all=False*, *almost_all=False*)

Iterator yielding information about path (defaults to current directory)

Currently this will only list the contents of a directory. More features will be added in the near future, but if there is a certain feature you are in need of, please don't hesitate to submit an issue in our issue tracker or better yet submit a pull request

**Parameters**

- **path** (*str*) – The directory to list
- **_all** (*boolean*) – If True files starting with "." are not ignored
- **almost_all** (*boolean*) – Like _all, but do not include "." and ".."

unitils.**mv**(*src*, *dst*)

Move or rename src to dst.

**Parameters**

- **src** (*str*) – The file/directory to move
- **dst** – The destination for the files to be moved
- **dst** – str

`unitils.`**`system_call`**(*command*, *stdin=-1*, *stdout=-1*, *stderr=-2*, *shell=False*)

Helper function to shell out commands. This should be platform agnostic.

Arguments are the same as to subprocess.Popen. Returns (stdout, stderr, returncode)

`unitils.`**`watch`**(*command*, *interval=2*)

Iterator yielding a tuple of (stdout, stderr, returncode) returned by issuing command to the system repeatedly. By default sleeps for 2 seconds between issuing commands.

**Parameters**

- **`command`** (*`str, list`*) – The command to issue to the system
- **`interval`** (*`int`*) – The number of seconds to wait before issuing command again

**Returns** Iterator of three-tuples containing (stdout, stderr, returncode)

`unitils.`**`wc`**(*files*, *lines=False*, *byte_count=False*, *chars=False*, *words=False*, *max_line_length=False*)

Yields newline, word and byte counts for each file and a total line if more than one file is specified

**Parameters**

- **`files`** (*`iterable`*) – An iterable of open, file-like objects or strings containing filenames
- **`lines`** (*`boolean`*) – Whether to include line counts
- **`byte_count`** (*`boolean`*) – Whether to include bytes count
- **`chars`** (*`boolean`*) – Whether to include chars count
- **`words`** (*`boolean`*) – Whether to include word count
- **`max_line_lenth`** (*`boolean`*) – Whether to include max_line_length

`unitils.`**`which`**(*cmd*, *_all=False*)

Search the PATH for the first occurance of cmd.

**Parameters**

- **`cmd`** (*`str`*) – The command for which to search
- **`_all`** (*`boolean`*) – If True, all occurances of cmd on the PATH will be returned

# Python Module Index

# Index