RATOM Documentation

Release 3.0.0

qtfkwk

Oct 24, 2018

CONTENTS

CHAPTER

ONE

OVERVIEW

1.1 Description

RATOM stands for "Rage Against The Outdated Machine".

Its purpose is to simply update all the things that need updating.

The primary use for RATOM is under current Python 2.x on a supported operating system that uses one or more of the supported software.

1.2 Features

- Supports macOS, FreeBSD (freebsd-update, portsnap, pkg), Debian and derivatives (apt-get), Red Hat and derivatives (yum), ClamAV/freshclam, Homebrew, Cask, Perlbrew, CPAN Minus (cpanm), pyenv, pip, rbenv, gem, npm, Metasploit Framework, Git repositories, and Microsoft AutoUpdate via a plugin architecture
- Markdown-formatted output with all update and informational commands shown with their output and in pretty terminal colors via the blessings package; also allows subsequent processing by redirecting or piping
- Dry run mode (`-n`) processes configuration file and command line arguments, performs checks and intermediate processing, prints commands to show what will run given configuration and system settings, but doesn't actually update anything
- Configuration via `~/.ratom/config.json` or an argument to `-c` option; allows switching the ordering of plugins (not recommended), explicit enabling or disabling of plugins, and specifying a different path for the log file
- Logs intermediate processing commands and other informational and error messages to the configured log location (`~/.ratom/ratom.log` by default) or an argument to `-l` option
- Shows full configuration details if `--show-config` option is used; be sure to combine with `-n` if you don't want to update anything
- Each plugin provides a `check` function to determine whether to run and a `main` function that performs the update
- Full documentation in HTML (online, gzipped tar) and PDF (view, download) via Sphinx

1.3 Design

- Show all configuration, commands, and output
- Use a modular plugin architecture

- Generating a report should be easy
- Run sequentially to avoid issues
- Halt when a command fails

1.4 Installation

```
pip install ratom
```

Can also install from either the binary distribution (or "wheel") or source distribution files:

```
pip install ratom-3.0.0-py2-none-any.whl
pip install ratom-3.0.0.zip
```

1.5 Usage

```
usage: ratom [-h] [-n] [-c PATH] [-l PATH] [--show-config]
             [plugin [plugin ...]]

optional arguments:
  -h, --help     show this help message and exit

ratom options:
  -n             Dry run; don't actually update anything
  -c PATH        Use alternate configuration file; default:
                 ~/.ratom/config.json
  -l PATH        Log to PATH; default: ~/.ratom/ratom.log
  --show-config  Show full configuration details
  plugin         Specific plugin(s) to run in the specified order; default:
                 "macos freebsd aptget yum clamav geoip homebrew perlbrew
                 cpanm pyenv conda pip rbenv gem npm msf git macos_microsoft";
                 ignored if running a plugin directly
```

1.6 Examples

RATOM can be used in a few ways...

1. Install with pip and run via the installed `ratom` shim
2. Clone the Git repository or unzip the source distribution and run either `ratom/all.py` or one of the plugins directly
3. Do #2 but also add symlinks to somewhere in your PATH:

   ```
   cd ~/bin
   ln -s path/to/ratom/ratom/all.py ratom
   ```

4. Use the Python REPL (or programtically from other Python code). Import `ratom.all` or a specific plugin, then call a `main` function and pass any arguments in command-line fashion via the `argv` argument or a configuration dictionary via the `cfg` argument. See also the *API Reference*.

```
$ python
>>> import ratom.clamav
>>> ratom.clamav.check()
True
>>> ratom.clamav.main(['-n'])
...
>>> ratom.clamav.main()
...
>>> import ratom.all
>>> ratom.all.main()
...
```

1.7 Versions

- 3.0.0 (2018-10-24): Add support for Python 3
- 2.2.5 (2018-01-31): Convert releases table in documentation; UTC date in update hook
- 2.2.4 (2018-01-22): Remove extraneous newline in pip plugin
- 2.2.3 (2018-01-05): Fix cpanm plugin output; update usage in doc
- 2.2.2 (2018-01-05): Fix pyenv plugin to show latest anaconda versions
- 2.2.1 (2018-01-05): Activate geoip plugin
- 2.2.0 (2018-01-05): Add geoip plugin; fix conda in doc
- 2.1.5 (2018-01-02): Add conda self updating and don't confirm
- 2.1.4 (2017-12-30): Fix issue not building documentation
- 2.1.3 (2017-12-30): Fix conda and pip plugins output
- 2.1.2 (2017-12-30): Fix error "Upload failed (400): Only one sdist may be uploaded per release" in favor of zip versus tar.gz
- 2.1.1 (2017-12-30): pip plugin only upgrades packages not installed as dependencies and allows ignoring packages in the config file; fix –show-config output
- 2.1.0 (2017-12-29): Add conda plugin
- 2.0.12 (2017-12-18): Fix the rbenv plugin to properly show the latest stable 2.x version of Ruby
- 2.0.11 (2017-12-17): Fix bug in build system preventing the banner version from being updated
- 2.0.10 (2017-12-17): Fix the npm plugin to avoid "broken" outdated command; fix pyenv plugin to properly show the latest stable 2.x and 3.x versions of Python; other minor fixes
- 2.0.9 (2016-12-08): Removed cask plugin; remove the *–all* option from the *brew upgrade* command; fix the npm plugin; minor fixes to output
- 2.0.8 (2016-11-30): Added *–format=legacy* to *pip list* command in pip plugin; disable uploading documentation to pythonhosted site (will transition to readthedocs)
- 2.0.7 (2016-09-22): Use *–verbose* instead of *-v* with macos plugin *softwareupdate*
- 2.0.6 (2016-08-05): Added Contact section with URLs to header
- 2.0.5 (2016-08-05): Fix join error in cask plugin
- 2.0.4 (2016-08-03): Renamed macosx to macos & macosx_microsoft to macos_microsoft; increased verbosity of cask plugin; added banner to usage (-h)

- 2.0.3 (2016-08-02): Added ASCII art banner; improved logging in cask plugin
- 2.0.2 (2016-07-18): Fix cask plugin failing when *cask list* gives name along with " (!)"; fix clamav plugin failing due to exiting with 1 when already up-to-date
- 2.0.1 (2016-06-06): Fixes for freebsd plugin
- 2.0.0 (2016-06-05): Replaced pyenv global commands and fixed issue with vim in homebrew plugin; using os.path.realpath instead of readlink in git plugin; has function; added ckver, current, latest functions to freebsd plugin; using kron instead of date command; renamed microsoft plugin to macosx_microsoft; simplified function naming, logging; updated documentation
- 1.1.0 (2016-05-26): Changed UnknownModule exception to UnknownPlugin
- 1.0.7 (2016-05-26): Added descriptions of aptget and yum plugins to Plugins section of documentation
- 1.0.6 (2016-05-26): Add aptget and yum plugins to documentation
- 1.0.5 (2016-05-26): Pipe stderr in runp, aptget and yum plugins
- 1.0.4 (2016-05-26): Documentation: moved content from readme, fixed typo, renamed apple plugin to macosx; Code: run brew upgrade via shell, log exceptions as errors, log command
- 1.0.3 (2016-05-25): Improved release automation
- 1.0.2 (2016-05-25): More work on release script and documentation
- 1.0.1 (2016-05-25): Fixed release script, rearranged documentation
- 1.0.0 (2016-05-25): Initial release

1.8 Issues

Please report issues via Github Issues.

Better yet, fork the Github repository, fix the issue, and send a PR (pull request)!

1.9 Contact

- Github
- PyPI
- Documentation

1.10 To do

- update Perl modules via CPANM for all perlbrew perls?
- update Python modules via pip for all pyenv pythons?
- update Ruby gems for all rbenv rubys?

CHAPTER

TWO

PLUGINS

The subsections below list details about each individual plugin.

In general, it is the user's responsibility to handle various side effects of individual plugins, for example some plugins may require reprocessing terminal startup scripts (`.bashrc`, etc) or even rebooting. Reprocessing startup scripts can be acheived by restarting the terminal session either by issuing `exit` or Ctrl+D and reopening a terminal or logging in again, or perhaps running `exec $SHELL`.

Also note that RATOM runs as the user that runs it, upon the assumption that the user has the appropriate permissions, etc. Of course, if a plugin passes its `check` function, but lacks permissions to perform the update then the command *should* fail, but this depends on the individual update utility. If it fails (exits with a non-zero value), RATOM will halt. If this occurs, you might have an issue of this kind, and your courses of action include fixing permissions of the item and its files for your user, disabling the plugin in the configuration file, or modifying the plugin's `check` or `main` functions to work correctly. Some ideas for the last fix might be to check if the user has proper permissions, has a particular UID/EUID/GID/EGID, or to run the command(s) via `su` or `sudo`.

2.1 all

Attempts to run all plugins listed as command line arguments, in the plugins list in the configuration file, or in the plugins list in the `common.defaults` dictionary (`common.defaults['plugins']`). Regardless where the list of plugins is found, the plugins are run in the order given. The default order is designed to update operating systems first, then any other security-related items, followed by development tools and personal tools/repositories, and finally any GUI-based update mechanisms. Of course, each plugin must also pass its respective `check` function in order to actually perform the update. This process prevents blindly attempting to run plugins on systems that either don't have the software they update or more importantly, when the user doesn't want RATOM to update them.

2.2 aptget

Updates Debian or Debian-based system (Ubuntu, Kali...) via `apt-get update`, `apt-get dist-upgrade -y`, then removes unnecessary packages via `apt-get autoremove -y`.

2.3 clamav

Manually updates Clam AntiVirus signatures via `freshclam`. This is in contrast to using the `freshclamd` daemon which can likely do a better job of keeping the signatures up-to-date. However, running freshclam manually confirms that the signatures are up-to-date whether the system uses the daemon or not.

2.4 conda

Updates conda and Anaconda packages via `conda update -yn base conda` and `conda update -y --all`.

2.5 cpanm

Uses cpan-outdated, which is installable via `cpanm App::cpanoutdated`, to check for outdated Perl/CPAN modules (`cpan-outdated -p`), then updates each via `cpanm`. This plugin runs against the "current" Perl, without regard for or knowledge of things like Perlbrew.

2.6 freebsd

Actually attempts to update several individual FreeBSD-specific items as a single plugin. Supported items are freebsd-update, portsnap, and pkg. This plugin only updates the currently-tracked branch of FreeBSD; it does not upgrade your system to the current release branch; i.e. if your system has 10.2-RELEASE and 10.3-RELEASE is available, it will not upgrade to 10.3-RELEASE for you, but it will tell you.

2.7 gem

Runs `gem update` to update globally-installed gems for the "current" selected Ruby, without regard for or knowledge of things like rbenv.

2.8 geoip

Runs `geoipupdate -v` to update the GeoIP database according to the configuration.

2.9 git

Performs a `git pull` for each repository or symlink to a repository in `~/.ratom/git/` after first showing the remote upstream repository via `git remote -v`. The `check` function fails if either the `~/.ratom/git` directory does not exist or it does not contain any symlinks or directories. Each repository path is *expanded* via `os.path.realpath` (doc) to operate directly against the canonical path.

2.10 homebrew

Updates Homebrew via `brew update; brew upgrade --all`, then performs clean up via `brew cleanup`.

It also attempts to avoid a specific issue discussed here encountered when upgrading Vim (and using pyenv) by temporarily restoring the "system" version of Python via the PYENV_VERSION environment variable before running the upgrade command. Note that this has initial success but should still be considered a work-in-progress.

2.11 macos

Updates macOS via the `softwareupdate` utility. An update may require reboot and the output will indicate this; the rest of the update process will continue and it is the user's responsibility to perform the reboot.

2.12 macos_microsoft

Runs the GUI-based Microsoft AutoUpdate utility, which updates Microsoft software installed on a macOS system. Unfortunately, this appears to be the only way to confirm that the software is up-to-date, since searching for a command-line utility has so far been fruitless. This plugin blocks while the user clicks the "Check for Updates" button, installs any updates, then closes the GUI.

2.13 msf

Updates Metasploit Framework via `msfupdate`.

2.14 npm

Checks for updates, attempts to update, and confirms updates of global NPM (Node.js) modules.

2.15 perlbrew

Updates Perlbrew and shows the installed versions of Perl and available versions of Perl; does not install any version of Perl for you.

2.16 pip

Upgrades the `pip` package first, then attempts upgrading all other installed packages.

2.17 pyenv

Shows the installed versions of Python and the latest versions in the 2.7 and 3.5 branches; does not install any version of Python for you.

2.18 rbenv

Show the installed versions of Ruby and the latest version in the 2.3 branch; does not install any version of Ruby for you.

2.19 yum

Updates Red Hat or a derivative (Fedora, CentOS...) via `yum update -y`.

CHAPTER

THREE

API REFERENCE

3.1 ratom.common

Common things shared across RATOM

exception ratom.common.**CommandFailed**
command exited with non-zero return code

exception ratom.common.**Error**
general error exception

exception ratom.common.**IntermediateCommandFailed**
intermediate command exited with non-zero return code

exception ratom.common.**UnknownPlugin**
encountered an unknown plugin

ratom.common.**error**(*msg*)
print error message to log if we are logging

ratom.common.**fetch**(*uri*, *params=None*, *soup=True*)
fetch data from a web URI via requests and return a BeautifulSoup object or the data if soup is False

ratom.common.**has**(**commands*)
test if command(s) are in PATH

ratom.common.**header**(*r*, *c*, *cfg*, *show_config=False*)
print the header

- `r`: running configuration dictionary
- `c`: configuration file path
- `cfg`: configuration dictionary from the configuration file
- `show_config`: shows full configuration details if true

ratom.common.**info**(*msg*)
print informational message to log if we are logging

ratom.common.**init**(*argv=None*, *cfg=None*)
process the arguments and configuration file, set up logging, and print the header

- `argv`: passed to `parse_args` method of `argparse.ArgumentParser` instance; uses `sys.argv` if None
- `cfg`: avoids rerunning if `cfg` is already defined

ratom.common.**replace**(*replacements*, *s*)
make all replacements in a string

ratom.common.**run**(*c*, *prompt='$ '*, *dryrun=False*, *shell=False*, *good=0*)
print and run one or more commands

- c: command or list of commands
- prompt: prompt to display when printing the command
- dryrun: just prints the command if true
- shell: passed to run_
- good: allowed exit codes; single integer or list of integers

ratom.common.**run_**(*c*, *shell=False*, *good=0*)
just run a command

- c: command
- shell: run via shell if true; avoid when possible, but necessary for things like * expansion
- good: allowed exit codes; single integer or list of integers

ratom.common.**runp**(*c*, *prompt='$ '*, *dryrun=False*, *shell=False*, *check=False*, *verbose=False*)
run a command and return the exit code, stdout and stderr back to the caller

- c: command
- prompt: prompt to display when printing the command
- dryrun: just prints the command if true
- shell: passed to run_
- check: don't raise an exception if true; for use only by check functions
- verbose: print command and output if true

ratom.common.**section**(*n*, *c*, *dryrun=False*, *good=0*)
shorthand for a simple section

- n: name
- c: command or list of commands
- dryrun: passed to run function
- good: allowed exit codes; single integer or list of integers

ratom.common.**section_begin**(*m*, *a=''*, *backticks=True*, *prefix='##'*)
begin a section in the standard way

- m: header text
- a: additional content
- backticks: prints backticks for beginning a code block
- prefix: override the default header prefix

ratom.common.**section_end**(*backticks=True*)
end a section in the standard way

3.2 Plugins

3.2.1 ratom.all

imports and runs all plugins

ratom.all.**main**(*argv=None*, *cfg=None*)
runs all plugins

3.2.2 ratom.aptget

update Debian via apt-get

ratom.aptget.**check**()
check if can update Debian via apt-get

ratom.aptget.**main**(*argv=None*, *cfg=None*)
update Debian via apt-get

3.2.3 ratom.clamav

update ClamAV signatures

ratom.clamav.**check**()
check if can update ClamAV signatures

ratom.clamav.**main**(*argv=None*, *cfg=None*)
update ClamAV signatures

3.2.4 ratom.conda

update Anaconda packages via conda

ratom.conda.**check**()
check if can update Anaconda packages via conda

ratom.conda.**main**(*argv=None*, *cfg=None*)
update Anaconda packages via conda

3.2.5 ratom.cpanm

update Perl modules via CPANM

ratom.cpanm.**check**()
check if can update Perl modules via CPANM

ratom.cpanm.**main**(*argv=None*, *cfg=None*)
update Perl modules via CPANM

3.2.6 ratom.freebsd

update FreeBSD

ratom.freebsd.**check**()
check if can update FreeBSD

ratom.freebsd.**main**(*argv=None*, *cfg=None*)
update FreeBSD

3.2.7 ratom.gem

update Ruby gems

ratom.gem.**check**()
check if can update Ruby gems

ratom.gem.**main**(*argv=None*, *cfg=None*)
update Ruby gems

3.2.8 ratom.geoip

update GeoIP database

ratom.geoip.**check**()
check if can update GeoIP database

ratom.geoip.**main**(*argv=None*, *cfg=None*)
update GeoIP database

3.2.9 ratom.git

update Git repositories

ratom.git.**check**(*p*)
check if can update Git repositories

ratom.git.**main**(*argv=None*, *cfg=None*)
update Git repositories

3.2.10 ratom.homebrew

update Homebrew packages

ratom.homebrew.**check**()
check if can update Homebrew packages

ratom.homebrew.**main**(*argv=None*, *cfg=None*)
update Homebrew packages

3.2.11 ratom.macos

update macOS

ratom.macos.**check**()
check if can update macOS

ratom.macos.**main**(*argv=None*, *cfg=None*)
update macOS

3.2.12 ratom.macos_microsoft

update Microsoft software on macOS

ratom.macos_microsoft.**check**()
check if can update Microsoft software on macOS

ratom.macos_microsoft.**main**(*argv=None*, *cfg=None*)
update Microsoft software on macOS

3.2.13 ratom.msf

update Metasploit Framework

ratom.msf.**check**()
check if can update Metasploit Framework

ratom.msf.**main**(*argv=None*, *cfg=None*)
update Metasploit Framework

3.2.14 ratom.npm

update global NPM modules

ratom.npm.**check**()
check if can update global NPM modules

ratom.npm.**main**(*argv=None*, *cfg=None*)
update global NPM modules

3.2.15 ratom.perlbrew

update Perlbrew and check for updated Perl

ratom.perlbrew.**check**()
check if can update Perlbrew

ratom.perlbrew.**main**(*argv=None*, *cfg=None*)
update Perlbrew and check for updated Perl

3.2.16 ratom.pip

update Python packages via pip

`ratom.pip.`**`check`**`()`
check if can update Python packages via pip

`ratom.pip.`**`main`**`(`*argv=None*, *cfg=None*`)`
update Python packages via pip

3.2.17 ratom.pyenv

check for new Python versions in pyenv

`ratom.pyenv.`**`check`**`()`
check if can check for new Python versions in pyenv

`ratom.pyenv.`**`main`**`(`*argv=None*, *cfg=None*`)`
check for new Python versions in pyenv

3.2.18 ratom.rbenv

check for new Ruby versions in rbenv

`ratom.rbenv.`**`check`**`()`
check if can check for new Ruby versions in rbenv

`ratom.rbenv.`**`main`**`(`*argv=None*, *cfg=None*`)`
check for new Ruby versions in rbenv

3.2.19 ratom.yum

update Red Hat via yum

`ratom.yum.`**`check`**`()`
check if can update Red Hat via yum

`ratom.yum.`**`main`**`(`*argv=None*, *cfg=None*`)`
update Red Hat via yum

PYTHON MODULE INDEX

r

INDEX

S

U