RATOM Documentation

Release 1.0.0

qtfkwk

May 25, 2016

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RATOM stands for "Rage Against The Outdated Machine".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 Mac OSX, FreeBSD (freebsd-update, portsnap, pkg), ClamAV/freshclam, Homebrew, Cask, Perlbrew, CPAN Minus, 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 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
- Each plugin provides a `check` function to determine whether to run and a `main` function that performs the update
- Full documentation online, or as HTML (gzipped tar) or PDF (view, download) via Sphinx

1.3 Design

- Show all configuration, commands, and output
  - intermediate commands are shown in the log versus the output for brevity's sake

- Use a modular plugin architecture, not only for code organization, but to easily support more software
- 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-1.0.0-py2-none-any.whl
pip install ratom-1.0.0.zip
```

1.5 Usage

```
usage: all.py [-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:
                 "macosx freebsd clamav homebrew cask perlbrew cpanm pyenv pip
                 rbenv gem npm msf git 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. Note that if you want to call a `check` function, you'll need to import `raton.common`. See also the *API Reference*.

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

1.7 Versions

Version	Date	Comments
1.0.0	2016-05-25	Initial release
1.0.1	2016-05-25	Fixed release script, rearranged documentation

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

- support Debian/Ubuntu (apt-get), Red Hat/Fedora/CentOS (yum)...
- run `brew upgrade --all` with the pyenv version set to 'system' without using `pyenv global`
- 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 apple

Updates Apple Mac OSX 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.

2.3 cask

Updates Homebrew casks by running `brew cask info` for each installed cask package as an intermediate command to determine whether there is an update available. If so, it runs `brew cask install` to install the updated cask. Finally, `brew cask cleanup` is run to remove temporary files and perform general maintainance tasks.

2.4 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.

2.5 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, pkg, and a custom utility called ckver, that queries the freebsd.org website to compare the latest release version to the current running version on the system. 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 ckver will tell you if a new release is avaiable.

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 git

Performs a `git pull` for each repository or symlink to a repository in `~/.ratom/git/` after first showing where the symlink points (if it's a symlink) and the set of tracked repositories via `git remote -v`. The `check` function fails if either the `~/.ratom/git` directory does not exist or it does not contain any repositories.

2.9 homebrew

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

It also attempts to avoid specific issues encountered when upgrading vim by restoring the "system" version of Python via pyenv before running the upgrade command. This has had mixed success, has some unintentional temporary system-wide side effects, and should be considered a work-in-progress.

2.10 microsoft

Runs the GUI-based Microsoft AutoUpdate utility, which updates Microsoft software installed on a Mac OSX 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.

2.11 msf

Updates Metasploit Framework via `msfupdate`.

2.12 npm

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

2.13 perlbrew

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

2.14 pip

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

2.15 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.

2.16 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.

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.**UnknownModule**
encountered an unknown module

ratom.common.**args**(*argv=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

ratom.common.**begin**(*m*, *a=''*)
begin a section in the standard way

- m: header text
- a: additional content

ratom.common.**end**()
end a section in the standard way

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.**run**(*c*, *prompt='$ '*, *dryrun=False*)
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

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

- `c`: command
- `shell`: run via shell if true; avoid when possible, but necessary for things like * expansion

`ratom.common.`**`runp`**(*c*, *check=False*)
run a command and return the exit code, stdout and stderr back to the caller

- `c`: command
- `check`: don't raise an exception if true; for use only by `check` functions

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

- `n`: name
- `c`: command or list of commands
- `dryrun`: passed to `run` function

3.2 ratom.all

imports and runs all plugins

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

3.3 Plugins

3.3.1 ratom.cask

update Cask packages

`ratom.cask.`**`check`**()
check if can update Cask packages

`ratom.cask.`**`main`**(*argv=None*, *cfg=None*)
update Cask packages

3.3.2 ratom.clamav

update ClamAV signatures

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

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

3.3.3 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

3.3.4 ratom.freebsd

update FreeBSD

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

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

3.3.5 ratom.gem

update Ruby gems

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

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

3.3.6 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

3.3.7 ratom.homebrew

update Homebrew packages

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

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

3.3.8 ratom.macosx

update Mac OSX

ratom.macosx.**check**()
check if can update Mac OSX

ratom.macosx.**main**(*argv=None*, *cfg=None*)
update Mac OSX

3.3.9 ratom.microsoft

update Microsoft software on Mac OSX

ratom.microsoft.**check**()
check if can update Microsoft software on Mac OSX

ratom.microsoft.**main**(*argv=None*, *cfg=None*)
update Microsoft software on Mac OSX

3.3.10 ratom.msf

update Metasploit Framework

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

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

3.3.11 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

3.3.12 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

3.3.13 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

3.3.14 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

3.3.15 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

r