redscheduler Documentation

Release 0.0.3-dev

Tyghe Vallard

July 10, 2015

CHAPTER 1

redscheduler

Use Redmine issues that model job specifications to easily setup jobs that need to be run. Then use the built in scraper to run them on any computer.

Learn more by viewing the full documentation here

1.1 TODO

Contents:

1.1.1 Installation

Virtualenv

Most likely you will want to install into a virtualenv to keep your code separate from the system's python packages.

This portion is completely optional

```
python setup.py bootstrap_virtualenv venv --prompt="(redscheduler)"
source venv/bin/activate
```

Install redscheduler

```
pip install -r requirements.txt
python setup.py install
```

For python 2.6 you will need to also install ordereddict

```
pip install ordereddict
```

1.1.2 Configuration

Configuration is split into two parts

- Local config file
- Redmine setup

Local Config File

You will need to ensure that you have setup your config file which needs to be located under your home directory as .redscheduler.config

If you are already familiar with the yaml format, then you will notice that the config file utilizes yaml. If you are not familiar with yaml format, then you may want to go read about that here

There is an example config file you can copy as your template in the repository called redscheduler.config.example that contains comments about what each configuration item means.

Options

siteurl This simply defines the url to your Redmine instance

Example: `https://www.foo.com`

apikey You can get this from the My Account page in Redmine

output_directory This is the top level directory where all Issue_#### directories are put.

When each job is run, it creates an Issue_#### directory inside this path and then is run within that created issue directory.

The issue directory can be referenced inside any job_def via:

```
{ISSUEDIR}
```

job_defs

The job_defs section simply lists all the jobs that you want to be able to run. Each job_def name needs to coorespond to a tracker name in Redmine. This is how issues are mapped to what job_def to run.

Each job_def has to define at least, `cli`, but here is a list of all additional configurable options.

cli This is the command that will be run for the job.

Only include the arguments that are static for this command and don't change between each time the command is run.

Note: Environmental variables are not replaced such as `$HOME` and `~/` so you will have to specify the absolute path to your executables and other files

Typically arguments that you always set should be specified in the cli and arguments that you sometimes change should be specified in the body of the issue you create.

Let's pretend you want to setup a redschedule job that will get a listing of files for a specific directory in long format. The command you would issue will always start with `ls -l`. The part that changes is the path to the directory you will want to get the listing for.

Here is the example job_def that we would use:

```
get_dir_listing:
    cli: "ls -l"
```

Then in the body of the issue you create in Redmine, you would put a directory path(or multiple, one per line). If the paths you put into the Redmine issue are:

```
/path/to/foo
/path/to/bar
```

Then redscheduler would run `ls -l /path/to/foo /path/to/bar`

cli Restrictions You cannot have more than one command in the cli field. That is, you cannot use shell operators such as `|`, `&&`, `>`, `;` to create complex pipelines.

If you need to create a more complex pipeline, then you will need to create a shell script that runs that pipeline and then specify that shell script in the cli Example:

> The goal is to be able to have your Redmine tracker issue specify what you want to grep out of /proc/cpus
>
> The shell pipeline would be:

```
cat /proc/cpus | grep
```

> That is, have the `cat` command send /proc/cpus information to standard out and then grep that output. We would put the search terms we want for grep in the Redmine issue that models this jobdef.
>
> Since you cannot have the `|` in the cli, you will need a shell script to do that for you

```
$> cat <<EOF > ~/catgrepcpus.sh
#!/usr/bin/env bash
cat /proc/cpus | grep $1
EOF
$> chmod 755 ~/catgrepcpus.sh
```

> Then configure the job_def as follows:

```
job_defs:
    pipeline_example:
        cli: /home/username/catgrepcpus.sh
```

> **Note**: Notice how we specify `/home/username/catgrepcpus.sh` and do not use `$HOME/catgrepcpus.sh` or `~/catgrepcpus.sh`. This is because of a a restriction in place by redscheduler that prevents it from expanding `~/` and bash variables in place to help keep things more secure.
>
> Now in the description of your redmine issue you would put the following to have it grep out the term `Physical`:

```
Physical
```

> Pretty simple!

stdout This is an optional option that specifies the location of the file where to place any output that gets generated on standard output from running the cli directive

Ommitting this option will use the default of `{ISSUEDIR}/stdout.txt`

stderr This is an optional option that specifies the location of the file where to place any output that gets generated on standard error from running the cli directive

Ommitting this option will use the default of `{ISSUEDIR}/stderr.txt`

uploads This is a list of files that you want to be uploaded to the issue once the job has completed. These paths need to include {ISSUEDIR} and cannot have a relative path that goes outside of the issue directory. That is the following will not work:

```
jobdefs:
    example:
        ...
        uploads:
            - /tmp/bob.txt
            - {ISSUEDIR}/../../etc/passwd
```

but the following would:

```
jobdefs:
    example:
        ...
        uploads:
            - {ISSUEDIR}/stdout.txt
            - {ISSUEDIR}/stderr.txt
            - {ISSUEDIR}/foo/myoutputfile.txt
```

Redmine Setup

The redmine setup is quite simple.

Job Def Tracker

Then you need to create a tracker with the same name as the job_defs you define.

This tracker **must** contain the following status workflow items:

- New
- In Progress
- Error
- Completed

Remine Job Issues

Think of each issue in Redmine as a Job instance that will be run. So if you make 100 issues in redmine, then 100 jobs will be run on the computer

Essentially the description portion of each issue defines the arguments that are combined with the local config's job_def's cli.

You can also attach files to the issue and reference them in the description of the issue like this `attachement:foo.txt`

Full Example

Assume we are using the example config(redscheduler.config.example) that comes with this project.

- output_directory is set to /tmp

We want to define a job that will convert comma separated files to tab separated files and then upload the result back to the issue when it is completed.

We know that in the shell you can simply use the `tr` command to do translate output by by substituting the tab `\t` character with a comma `,`. Here is the shell command that would do this:

```
tr ',' '\t' < /path/to/input.csv
```

Now that we know how the shell can do it, we will need to convert it to a job_def + Redmine issue.

First, what will remain the same every time we run the command?:

```
tr ',' '\t' <
```

We know from reading *CLI Example* that you cannot have the < in the job_def's cli field so we will need to make a shell script that we will use instead.

```
mkdir -p ~/jobdefscripts
cat <<EOF > ~/jobdefscripts/replace-comma-with-tab.sh
#!/usr/bin/env bash
tr ',' '\t' < \$1 > output.tsv
EOF
chmod 755 ~/jobdefscripts/replace-comma-with-tab.sh
```

You can see that the shell script is setup to direct the first argument given to it `$1` into the tr command and then direct the output into a file called output.tsv

Now, we can define our job in our ~/.redscheduler.config:

```
job_def:
    ...
    replace-comma-with-tab:
        cli: "/home/username/jobdefscripts/replace-comma-with-tab.sh"
        uploads:
            - {ISSUEDIR}/output.tsv
            - {ISSUEDIR}/stderr.txt
```

Make sure to create the tracker called `replace-comma-with-tab`

Now you can create jobs to run this task easily. You upload your files you want converted to the issue and reference them in the issue description.

Here would be an example description:

```
attachment:my.csv
```

Assuming, you have uploaded the attachment named my.csv to the issue when you created it.

Now, when the job runs, you can expect the following to happen

1. `/tmp/Issue_#` is created for you(where # is replaced with the issue id)
2. `/tmp/Issue_#/output.tsv` and `/tmp/Issue_1/stderr.txt` are created with output from running your command

   In this case, stdout.txt should be empty as you are redirecting output
   into `output.tsv`
   If your command generates errors, you will get output in `stderr.txt`

3. `/tmp/Issue_#/output.tsv` is created with the output from your command.

4. Additionally, your command is run within the `/tmp/Issue_#` directory.

   If any relative paths are used for files, they will be relative to that directory

5. When the task runs it will change the issue status from `New` to `In Progress` to `Completed` or `Error`.
6. When the task completes it will upload output.tsv and stderr.txt back to the issue

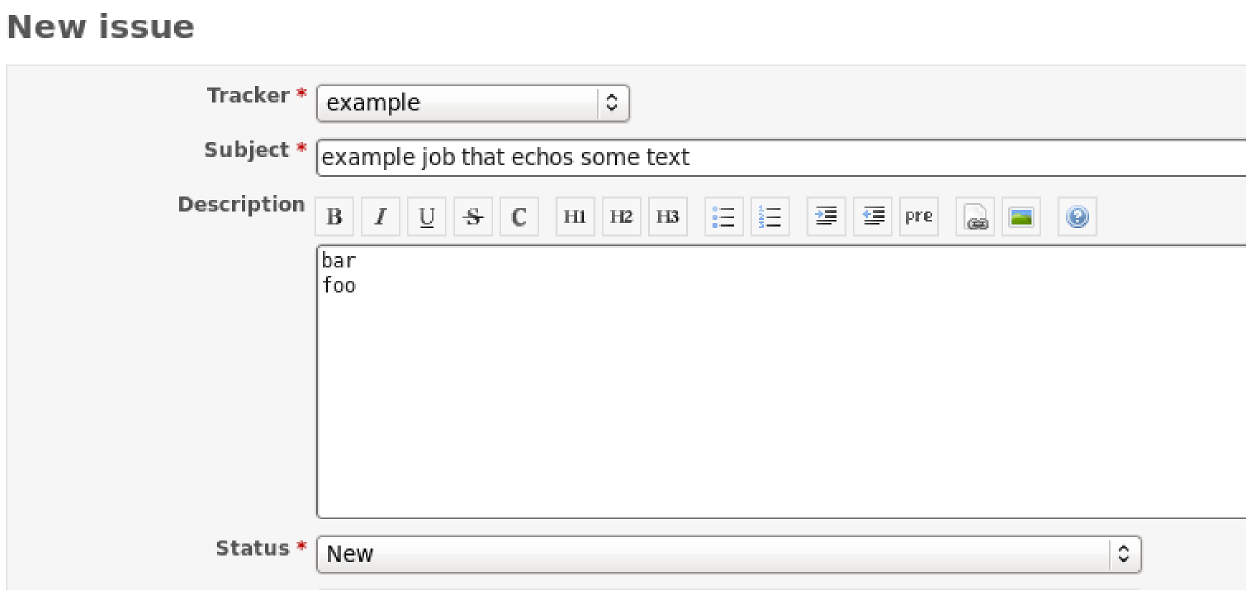
1.1.3 Running Issues

Once you have configured both your Redmine trackers and the config file to match them you are set up to start creating issues and running them.

You can easily run your issues using the `runissue` command which simply fetches the issue id you give it and then executes the defined cli in the config file mixed with arguments from the issue's description.

Using the default configuration file `redscheduler.config.example` that comes with the project when you download it, we will do an example.

Create your first issue



New issue

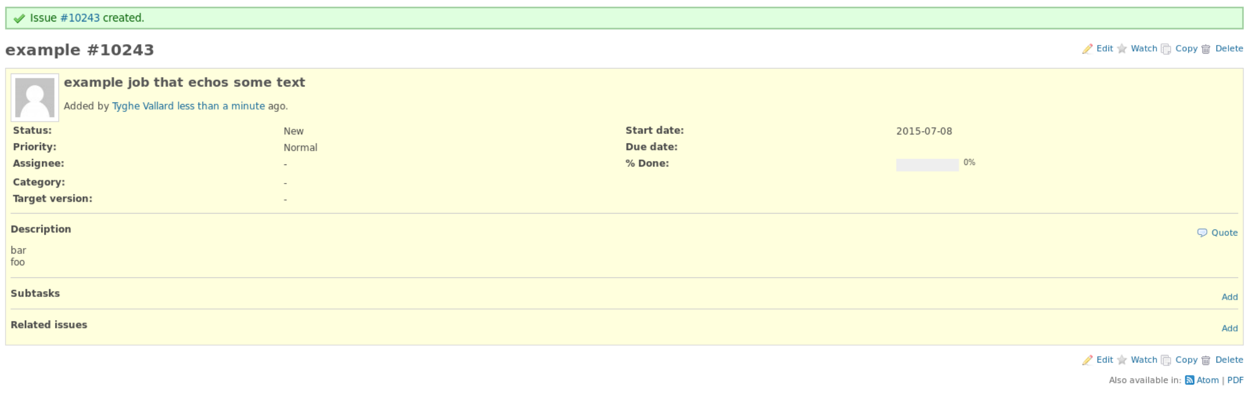
Tracker *: example

Subject *: example job that echos some text

Description:

```
bar
foo
```

Status *: New



Issue #10243 created.

example #10243

example job that echos some text

Added by Tyghe Vallard less than a minute ago.

Status:	New	Start date:	2015-07-08
Priority:	Normal	Due date:	
Assignee:	-	% Done:	0%
Category:	-		
Target version:	-		

Description

bar
foo

Subtasks

Related issues

Also available in: Atom | PDF

Here we can see that issue #10243 was created. We will need this number to run the issue.

Run the issue

```
$> runissue 10243
Updated issue status with output location and set to In Progress
Creating /tmp/Issue_10243 to run in
Running echo -e foo
bar bar foo
Return code was 0
Saving issue with new status Completed
```

Check output file contents

You can see that the issue wrote all standard output into the defined `stdout` file, output.stdout.

```
$> cat /tmp/Issue_10243/output.stdout
foo
bar bar foo
```

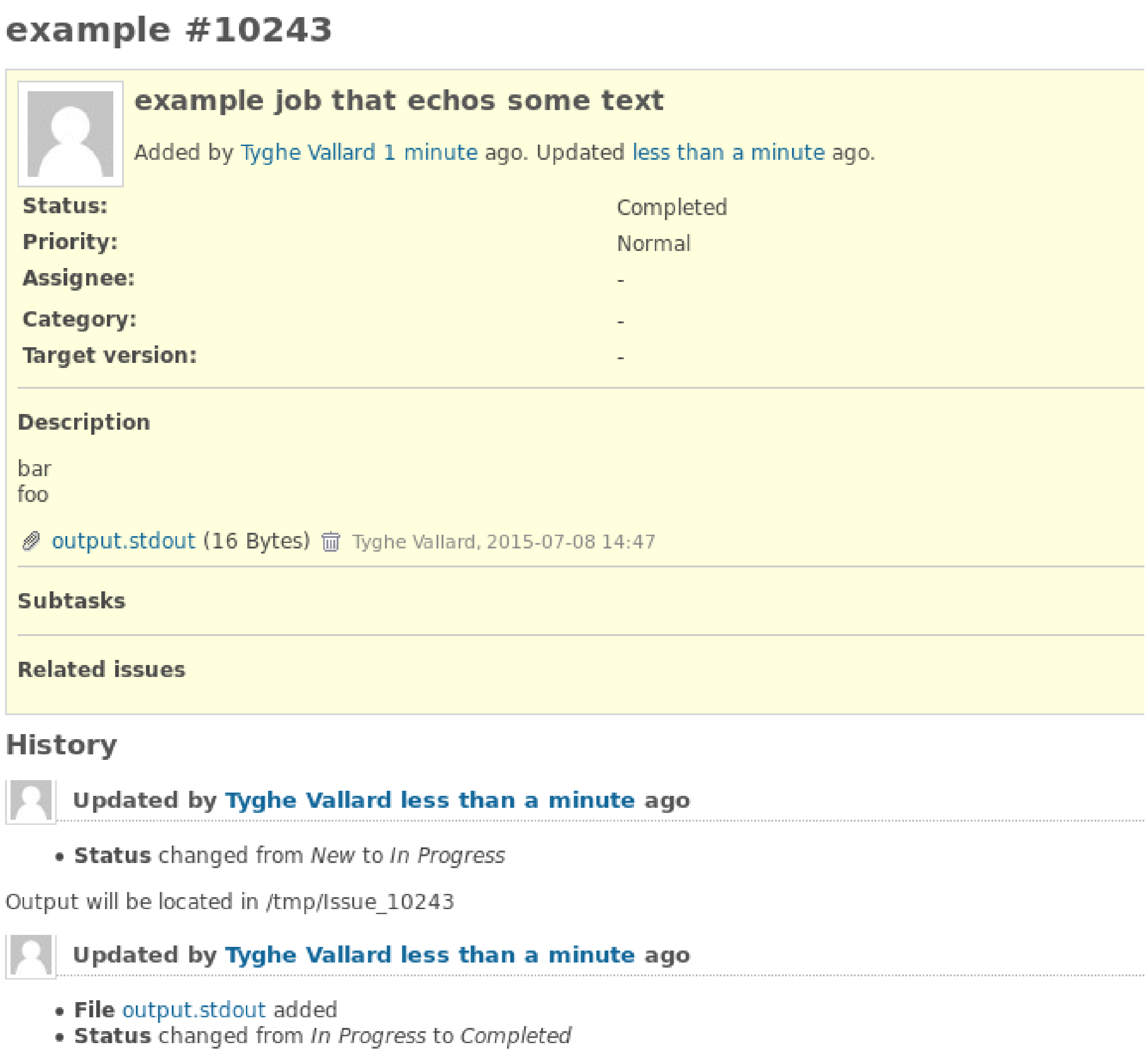
Additionally, stderr was empty(`Size:  0`)

```
$> stat /tmp/Issue_10243/stderr.txt
  File: `/tmp/Issue_10243/stderr.txt'
  Size: 0           Blocks: 0          IO Block: 4096   regular empty file
Device: 803h/2051d  Inode: 356         Links: 1
Access: (0644/-rw-r--r--)  Uid: (500/username)   Gid: (500/username)
Access: 2015-07-08 14:49:24.369910594 -0400
Modify: 2015-07-08 14:49:24.369910594 -0400
Change: 2015-07-08 14:49:24.369910594 -0400
```

Verify issue

Here you can see that the issue was first updated to show that it was running by changing the status from `New` to `In Progress`. It also creates a note showing the output location that the job will be run in(`/tmp/Issue_10243`)

After the job completes it updated the issue again, changing the status from `In Progress` to `Completed`. Also, because our config has uploads defined, the output.stdout file was uploaded back to the issue.



example #10243

example job that echos some text

Added by Tyghe Vallard 1 minute ago. Updated less than a minute ago.

Status:	Completed
Priority:	Normal
Assignee:	-
Category:	-
Target version:	-

Description

bar
foo

output.stdout (16 Bytes) Tyghe Vallard, 2015-07-08 14:47

Subtasks

Related issues

History

Updated by Tyghe Vallard less than a minute ago

- **Status** changed from *New* to *In Progress*

Output will be located in /tmp/Issue_10243

Updated by Tyghe Vallard less than a minute ago

- **File** output.stdout added
- **Status** changed from *In Progress* to *Completed*

CHAPTER 2

Indices and tables

- genindex
- modindex
- search