did

Apr 19, 2021

Contents

This is **did**, a command line tool which will help you to easily gather data for your weekly, monthly or yearly reports. It suports plugins for various data sources such as git, trac, wiki, bugzilla and more. It can be used to gather stats for a single user as well as for the whole team and merging them together. It saves you from that boring stuff of making notes and later putting them together.

CHAPTER 1

Table of Contents

1.1 did

What did you do last week, month, year?

1.1.1 Description

Comfortably gather status report data (e.g. list of committed changes) for given week, month, quarter, year or selected date range. By default all available stats for this week are reported.

Based on the config, `did` explores user's activity for given tools in provided time frame. For example checks all configured git repositories for the list of commits or contacts Bugzilla to search for bugs created, modified or closed.

Some information (like git commits) is gathered from the local file system, but usually individual plugins are contacting remote API of the tool to query for the latest data. For tools which provide a public API there is no need for authentication. Some plugins support Kerberos, other need to create an authentication token. See individual plugin documentation for details.

1.1.2 Synopsis

Usage is straightforward:

```
did [this|last] [week|month|quarter|year] [opts]
```

1.1.3 Examples

Gather all stats for current week:

```
did
```

Show me all stats for today, yesterday, last Friday:

```
did today
did yesterday
did last friday
```

Gather stats for the last month:

```
did last month
```

See `did --help` for complete list of available stats.

1.1.4 Options

The list of available options depends on which plugins are configured. Here's the list of general options which are not related to any plugin:

Select

At least one email address needs to be provided on command line unless defined in the config file. Use the complete email address format `Name Surname <email@example.org>` to display full name in the report output. For date values `today` and `yesterday` can be used instead of the full date format.

--email=EMAILS User email address(es)

--since=SINCE Start date in the YYYY-MM-DD format

--until=UNTIL End date in the YYYY-MM-DD format

Format

The default output is plain text of maximum width 79 characters. This can be adjusted using the `--width` parameter. To disable shortening altogether use `--width=0`. The default width value can be saved in the config file as well. Use `--format=wiki` to enable simple MoinMoin wiki syntax. For stats which support them, `--brief` and `--verbose` can be used to specify a different level of detail to be shown.

--format=FMT Output style, possible values: text (default) or wiki

--width=WIDTH Maximum width of the report output (default: 79)

--brief Show brief summary only, do not list individual items

--verbose Include more details (like modified git directories)

Utils

Multiple emails can be used to put together a team report or to gather stats for all of your email aliases. For this use case `--total` and `--merge` can be used to append the overall summary at the end or merge all results into a single report respectively. Use `--debug` or set the environment variable `DEBUG` to 1 through 5 to set the desired level of debugging.

--config=FILE Use alternate configuration file (default: 'config')

--total Append total stats after listing individual users

--merge Merge stats of all users into a single report

--debug Turn on debugging output, do not catch exceptions

See `did --help` for complete list of available options.

1.1.5 Install

Install directly from Fedora/Copr repository:

```
yum install did
```

Or use pip to install from Python Package Index:

```
pip install did
```

You may want to install some or all extra requires:

```
pip install did[plugin]
pip install did[all]
```

To build and execute in a docker container, run:

```
make run_docker
```

See documentation for more details about installation options.

1.1.6 Config

The config file `~/.did/config` is used to store both general settings and configuration of individual reports:

```
[general]
email = "Petr Šplíchal" <psplicha@redhat.com>
width = 79

[header]
type = header
highlights = Highlights
joy = Joy of the week ;-)

[tools]
type = git
did = /home/psss/git/did

[tests]
type = git
tests = /home/psss/git/tests/*

[trac]
type = trac
prefix = TT
url = https://some.trac.com/trac/project/rpc

[bz]
type = bugzilla
prefix = BZ
url = https://bugzilla.redhat.com/xmlrpc.cgi

[footer]
```

(continues on next page)

(continued from previous page)

```
type = footer
next = Plans, thoughts, ideas...
status = Status: Green | Yellow | Orange | Red
```

See plugin documentation for more detailed description of options available for particular plugin. You can also check python module documentation directly, e.g. `pydoc did.plugins.git` or use the example config provided in the package and web documentation.

1.1.7 Links

Git: https://github.com/psss/did

Docs: https://did.readthedocs.io

Issues: https://github.com/psss/did/issues

Releases: https://github.com/psss/did/releases

Copr: https://copr.fedoraproject.org/coprs/psss/did

PIP: https://pypi.org/project/did

1.1.8 Authors

Petr Šplíchal, Karel Šrot, Lukáš Zachar, Matěj Cepl, Ondřej Pták, Chris Ward, Tomáš Hofman, Martin Mágr, Stanislav Kozina, Paul Belanger, Eduard Trott, Martin Frodl, Randy Barlow, Alois Mahdal, Evgeni Golov, Stanislav Ochotnický, Maroš Kopec, Robbie Harwood, Christopher Sams, Thomas Heute, Giulio Fidente, Han Han, Qiao Zhao, Henrique Ferreiro, Jakub Vávra, Luigi Toscano, Lukáš Zapletal, Maryna Nalbandian, Dominika Hoďovská, Jakub Haruda, Han Han, Štěpán Němec and Evgeny Fedin.

1.1.9 Copyright

1.1.10 Status

1.2 Install

1.2.1 Fedora

In Fedora simply install the package:

```
dnf install did
```

That's it! :-)

1.2.2 Copr

Set up the did repository and install the tool using dnf:

```
dnf copr enable psss/did
dnf install did
```

This will bring dependencies for all core plugins as well.

1.2.3 PIP

Installing did using pip directly on the system is easy:

```
pip install did
```

Use virtual environments if you do not want to affect your system. Install virtualenv wrapper to make the work more comfortable:

```
sudo yum install python-virtualenvwrapper   # Fedora
sudo apt install virtualenvwrapper          # Ubuntu
```

Create a new virtual environment, upgrade tools, install did:

```
mkvirtualenv did
workon did
pip install --upgrade pip setuptools
pip install did
```

This installs the tool and basic requirements. Some of the plugins have additional dependencies. Use `did[plugin]` to install extra dependencies, for example:

```
pip install did[bugzilla]   # Install bugzilla deps
pip install did[docs]       # Get everything for building docs
pip install did[tests]      # And for testing
pip install did[all]        # Install all extra dependencies
```

Note: For plugins depending on gssapi (jira & rt) there are some extra dependencies:

```
sudo yum install gcc krb5-devel python-devel    # Fedora
sudo apt install gcc libkrb5-dev python-dev     # Ubuntu
```

See the pypi package index for detailed package information.

1.2.4 Docker

Please note: This is a first cut at doing a container version as a result; known issues:

- Kerberos auth may not be working correctly
- Container runs as privileged to access the conf file
- Output directory may not be quite right

This does not actually run the docker image as it makes more sense to run it directly. Use:

```
docker run --privileged --rm -it -v $(HOME)/.did:/did.conf $(USERNAME)/did
```

If you want to add it to your .bashrc use this:

```
alias did="docker run --privileged --rm -it -v $(HOME)/.did:/did.conf $(USERNAME)/did"
```

A couple of useful resources to get started with docker:

- https://fedoraproject.org/wiki/Docker
- https://fedoraproject.org/wiki/Getting_started_with_docker

1.3 Config

The config file `~/.did/config` is used to store both general settings and configuration of individual reports. Command line option `--config` allows to select a different config file from the config directory. This can serve as a kind of a profile and is especially useful for gathering team reports.

Use the `DID_DIR` environment variable to override the default config directory `~/.did` and use your custom location instead. For example if you prefer to keep you home directory clean you might want to add the following line into `.bashrc`:

```
export DID_DIR=~/.config/did/
```

1.3.1 General

Minimum config file should contain at least a `general` section with an email address which will be used for searching. Option `width` specifies the maximum width of the report, `quarter` can be used to choose a different start month of the quarter:

```
[general]
email = Petr Šplíchal <psplicha@redhat.com>
width = 79
quarter = 1
```

In order to load additional plugins from your custom locations provide paths to be searched in the `plugins` option:

```
[general]
email = Petr Šplíchal <psplicha@redhat.com>
plugins = ~/.did/plugins
```

Each path should be a package or module. This method works whether the package or module is on the filesystem or in an `.egg`.

1.3.2 Email

Use the full email format `Name Surname <login@example.org>` if you want to have your full name displayed in the output or choose the short one `login@example.org` if you don't care. Multiple email addresses can be provided, separated with a comma, in both config file and on the command line, for example:

```
did --email first@email.org,second@email.org
did --email first@email.org --email second@email.org
```

This can be useful if you have several email aliases or if you want to generate report for the whole team. Note that the full email address format can be used on the command line as well.

1.3.3 Aliases

Custom email or login alias can be provided in stats sections. This allows to override the default value for individual stats:

```
[github]
type = github
url = https://api.github.com/
login = psss
```

See *did.base.User* for detailed information about the advanced email/login alias support.

1.3.4 Order

Order of individual sections is based on the default order set for each plugin separately. You can adjust stats order by providing your desired value in respective config section, for example:

```
[tools]
type = git
order = 100
apps = /home/psss/git/apps
```

This would place the git stats at the top of your report, just after the header section. Check *Plugins* documentation for the default order information.

1.3.5 Example

Here's an example config file with all available plugins enabled. See *Plugins* documentation for more detailed description of options available for particular plugin. You can also check python module documentation, e.g. `pydoc did.plugins.git`.

```
[general]
email = Petr Splichal <psplicha@redhat.com>
width = 79

[header]
type = header
highlights = Highlights
joy = Joy of the week ;-)

[nitrate]
```

(continues on next page)

(continued from previous page)

```
type = nitrate

[bugzilla]
type = bugzilla
prefix = BZ
url = https://bugzilla.redhat.com/xmlrpc.cgi

[tools]
type = git
did = /home/psss/git/did
edd = /home/psss/git/edd
fmf = /home/psss/git/fmf

[github]
type = github
url = https://api.github.com/
login = psss

[gerrit]
type = gerrit
url = https://example.org/gerrit/
prefix = GR

[gitlab]
type = gitlab
url = https://gitlab.com/
token = <authentication-token>
login = <username>
ssl_verify = true

[pagure]
type = pagure
url = https://pagure.io/api/0/
login = <username>
token = <authentication-token>

[trac]
type = trac
prefix = TT
url = https://some.trac.com/trac/project/rpc

[trello]
type = trello
user = member

[rt]
type = rt
prefix = RT
url = https://tracker.org/rt/Search/Results.tsv

[jira]
type = jira
prefix = JIRA
project = ORG
url = https://issues.jboss.org/
ssl_verify = true
```

(continues on next page)

(continued from previous page)

```
[sentry]
type = sentry
url = https://sentry.io/api/0/
organization = team
token = ...

[wiki]
type = wiki
wiki test = http://moinmo.in/

[projects]
type = items
header = Work on projects
item1 = Project One
item2 = Project Two
item3 = Project Three

[footer]
type = footer
next = Plans, thoughts, ideas...
status = Status: Green | Yellow | Orange | Red
```

1.4 Examples

Let's have a look at a couple of real-life examples!

1.4.1 Config

I have created the following config file to track my work on tools development in git, bug updates in bugzilla, ticket updates in trac, plus my favorite header & footer I'm used to fill manually:

```
[general]
email = "Petr Šplíchal" <psplicha@redhat.com>
width = 79

[header]
type = header
high = Highlights
joy = Joy of the week ;-)

[tools]
type = git
did = /home/psss/git/did
edd = /home/psss/git/edd

[trac]
type = trac
prefix = TT
url = https://some.trac.com/trac/project/rpc

[bz]
type = bugzilla
prefix = BZ
```

(continues on next page)

(continued from previous page)

```
url = https://bugzilla.redhat.com/xmlrpc.cgi

[footer]
type = footer
next = Plans, thoughts, ideas...
status = Status: Green | Yellow | Orange | Red
```

1.4.2 Options

Here's how available command line options look like with this config. Note that `did` detects all enabled plugins and creates corresponding option groups for each of them:

```
usage: did [this|last] [week|month|quarter|year] [options]

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

Select:
  --email EMAILS   User email address(es)
  --since SINCE    Start date in the YYYY-MM-DD format
  --until UNTIL    End date in the YYYY-MM-DD format

Header:
  --header-high    Highlights
  --header-joy     Joy of the week
  --header         All above

Bugzilla stats:
  --bz-filed       Bugs filed
  --bz-patched     Bugs patched
  --bz-posted      Bugs posted
  --bz-fixed       Bugs fixed
  --bz-returned    Bugs returned
  --bz-verified    Bugs verified
  --bz-commented   Bugs commented
  --bz-closed      Bugs closed
  --bz             All above

Work on tools:
  --tools-did      Work on did
  --tools-edd      Work on edd
  --tools          All above

Tickets in trac:
  --trac-created   Tickets created in trac
  --trac-accepted  Tickets accepted in trac
  --trac-updated   Tickets updated in trac
  --trac-closed    Tickets closed in trac
  --trac           All above

Footer:
  --footer-next    Plans, thoughts, ideas...
  --footer-status  Status: Green | Yellow | Orange | Red
  --footer         All above
```

(continues on next page)

(continued from previous page)

```
Format:
  --format FORMAT  Output style, possible values: text (default) or wiki
  --width WIDTH    Maximum width of the report output (default: 79)
  --brief          Show brief summary only, do not list individual items
  --verbose        Include more details (like modified git directories)

Utils:
  --config FILE    Use alternate configuration file (default: 'config')
  --total          Append total stats after listing individual users
  --merge          Merge stats of all users into a single report
  --debug          Turn on debugging output, do not catch exceptions
```

1.4.3 Week

Now it's easy to find out what I was working on during this week:

```
> did
Status report for this week (2015-09-07 to 2015-09-13).

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Petr Šplíchal <psplicha@redhat.com>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

* Highlights

* Joy of the week

* Bugs fixed: 2
    * BZ#1261963 - wrong date format causes traceback
    * BZ#1248551 - status-report crashes when trac url is incorrect

* Work on did: 52 commits
    * 91ae8e7 - Enabled syntax highlighting for config example
    * 978add5 - Convert plugin order list into table
    * 5de5514 - Update welcome page and module documentation
    * 0773a3f - Handle invalid date format
    * 4deb67b - Handle invalid paths in the git plugin config
    * 2aace67 - Handle invalid url in trac plugin configuration
    * 717f9e4 - Consider ticket description change as update
    * e84e0fc - Allow turning off py.test output capture feature
    * 7ae7df1 - Check free command line arguments for typos
    * b4e110e - Include example config in docs, adjust man page
    * d623ef0 - Clarify a bit more did.cli.main() usage
    * 72aaa5d - Move module description to the module itself
    * ...

* Tickets updated in trac: 2
    * TT#0400 - Convert status-report to an open source project
    * TT#0490 - Add or improve missing test coverage for key use cases

* Plans, thoughts, ideas...

* Status: Green | Yellow | Orange | Red
```

1.4.4 Tools

I can check my work on tools development during the last month:

```
> did --tools last month
Status report for the last month (2015-08-01 to 2015-08-31).

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Petr Šplíchal <psplicha@redhat.com>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

* Work on did: 3 commits
    * 6167e4f - Adjustments after the stats refactoring
    * 3df5c60 - Include gerrit details as comments, fix exception
    * 6bc869f - Include 'items' plugin config example

* Work on edd: 13 commits
    * 77d5c94 - Bail out if no file selected with --list [fix #5]
    * eb4db1a - Document the Ctrl-Shift-V keyboard shortcut
    * 1888397 - Version bump and changelog entry for 0.2
    * 2f4b631 - Document new options, some adjustments
    * c18095c - New option --last, some reorganization [fix #1]
    * 437103e - Work around RHEL7 zenity bug [BZ#1060471]
    * 653c7de - Merge new option --list
    * dddbc85 - Use the primary mouse selection first [fix #2]
    * a025c1c - Packaging stuff, documentation update
    * 7b3e9c8 - Detect text editor if not set
    * a1a2b9a - Use 'txt' extension for the temporary file
    * dec9d63 - New option --shortcut for keyboard shortcut
    * 556d3c4 - Include a short usage message
```

1.4.5 Brief

It's also possible to list only a concise summary of each section using the `--brief` option or select only desired stats to be displayed. Special values `today` and `yesterday` can be used instead of typing the whole date string:

```
> did --bz-filed --bz-fixed --bz-verified --until today --brief
Status report for given date range (1993-01-01 to 2015-09-11).

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Petr Šplíchal <psplicha@redhat.com>
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Bugs filed: 845
* Bugs fixed: 427
* Bugs verified: 278
```

That's it! Now you can experiment yourself ;-)

1.5 Plugins

Modules in this directory are searched for available stats. Each plugin should contain a single class inheriting from StatsGroup. Stats from this group will be included in the report if enabled in user config. Name of the plugin should match config section type. Attribute `order` defines the order in the final report.

In addition to built-in plugins it is also possible to define your own stats. In order to enable such custom plugins add path to the python modules into the *General* section of the config file.

This is the default plugin order:

header	000
google	050
nitrate	100
bugzilla	200
git	300
github	330
gerrit	350
gitlab	380
pagure	390
trac	400
trello	450
rt	500
redmine	550
jira	600
sentry	650
wiki	700
items	800
footer	900

1.5.1 bugzilla

Bugzilla stats such as filed, fixed or verified bugs

This plugin uses `python-bugzilla` module to gather the stats. By default reports contain only publicly available issues. Use the `bugzilla login` command to initialize Bugzilla cookies or get an API key from the Preferences and store it in the config file `.config/python-bugzilla/bugzillarc`:

```
[bugzilla.redhat.com]
api_key=YOUR-API-KEY
```

Config example:

```
[bz]
type = bugzilla
prefix = BZ
url = https://bugzilla.redhat.com/xmlrpc.cgi
resolutions = notabug, duplicate
```

Resolutions: List of resolutions to be displayed at the end of the summary if bug is closed. By default `notabug` and `duplicate` are shown. Use `all` to always display resolution if available or `none` to turn off the feature completely.

Available options:

--bz-filed	Bugs filed
--bz-patched	Bugs patched
--bz-posted	Bugs posted
--bz-fixed	Bugs fixed

--bz-returned	Bugs returned
--bz-verified	Bugs verified
--bz-commented	Bugs commented
--bz-subscribed	Bugs subscribed
--bz-closed	Bugs closed
--bz	All above

class did.plugins.bugzilla.**Bug**(*bug*, *history*, *comments*, *parent*)
Bugzilla search

closed(*user*)
Moved to CLOSED and not later moved to ASSIGNED

commented(*user*)
True if comment was added in given time frame

fixed()
Moved to MODIFIED and not later moved to ASSIGNED

logs
Return relevant who-did-what pairs from the bug history

patched(*user*)
True if Patch was added to Keywords field by given user

posted()
True if bug was moved to POST in given time frame

returned(*user*)
Moved to ASSIGNED by given user (but not from NEW)

subscribed(*user*)
True if CC was added in given time frame

summary
Bug summary including resolution if enabled

verified()
True if bug was verified in given time frame

class did.plugins.bugzilla.**Bugzilla**(*parent*)
Bugzilla investigator

search(*query*, *options*)
Perform Bugzilla search

server
Connection to the server

class did.plugins.bugzilla.**BugzillaStats**(*option*, *name=None*, *parent=None*, *user=None*)
Bugzilla stats

order = 200

class did.plugins.bugzilla.**ClosedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs closed

Bugs which have been moved to the CLOSED state in given time frame and later have not been moved back to the ASSIGNED state (which would suggest the bug was not closed for a proper reason).

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**CommentedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs commented

All bugs commented by given user in requested time frame.

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**FiledBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs filed

Newly created bugs by given user, marked as the `Reporter`.

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**FixedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs fixed

Bugs which have been moved to the `MODIFIED` state in given time frame and later have not been moved back to the `ASSIGNED` state (which would suggest an incomplete fix).

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**PatchedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs patched

Gathers bugs with keyword `Patch` added by given user, denoting the patch for the issue is available (e.g. attached to the bug or pushed to a feature git branch).

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**PostedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs posted

Bugs with patches posted for review, detected by their status change to `POST` and given user set as `Assignee`.

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**ReturnedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs returned

Returned bugs are those which were returned by given user to the `ASSIGNED` status, meaning the fix for the issue is not correct or complete.

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**SubscribedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Bugs subscribed

All bugs subscribed by given user in requested time frame.

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.bugzilla.**VerifiedBugs**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Bugs verified

Bugs with `QA Contact` field set to given user or changed by the given user and having their status changed to `VERIFIED`.

fetch()
Fetch the stats (to be implemented by respective class).

1.5.2 confluence

Confluence stats such as created pages and comments

Configuration example (GSS authentication):

```
[confluence]
type = confluence
url = https://docs.jboss.org/
```

Configuration example (basic authentication):

```
[jboss]
type = confluence
url = https://docs.jboss.org/
auth_url = https://docs.jboss.org/rest/auth/latest/session
auth_type = basic
auth_username = username
auth_password = password
auth_password_file = ~/.did/confluence_password
```

Notes: * Optional parameter `ssl_verify` can be used to enable/disable

SSL verification (default: true)

- `auth_url` parameter is optional. If not provided, `url + "/step-auth-gss"` will be used for authentication.
- `auth_type` parameter is optional, default value is `gss`.
- `auth_username`, `auth_password` and `auth_password_file` are only valid for basic authentication, `auth_password` or `auth_password_file` must be provided, `auth_password` has a higher priority.

class did.plugins.confluence.**CommentAdded**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.confluence.**Confluence**
Confluence investigator

static search(*query*, *stats*, *expand=None*)
Perform page/comment search for given stats instance

class did.plugins.confluence.**ConfluenceComment**(*comment*)
Confluence comment results

class did.plugins.confluence.**ConfluencePage**(*page*)
Confluence page results

class did.plugins.confluence.**ConfluenceStats**(*option*, *name=None*, *parent=None*, *user=None*)
Confluence stats

order = 600

session
Initialize the session

class did.plugins.confluence.**PageCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Created pages

fetch()
Fetch the stats (to be implemented by respective class).

1.5.3 footer

Customizable footer

Config example:

```
[footer]
type = footer
next = Plans, thoughts, ideas...
status = Status: Green | Yellow | Orange | Red
```

class did.plugins.footer.**Footer**(*option*, *name=None*, *parent=None*, *user=None*)

order = 900

1.5.4 gerrit

Gerrit stats such as submitted, review or merged changes

Config example:

```
[gerrit]
type = gerrit
url = https://example.org/gerrit/
prefix = GR
# optional, True by default; set to False if the gerrit server
# does not support wip as search criteria.
wip = True
```

class did.plugins.gerrit.**AbandonedChanges**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)
Changes abandoned

fetch()
Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

class did.plugins.gerrit.**AddedPatches**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

Additional patches added to existing changes

fetch()

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

class did.plugins.gerrit.**Change**(*ticket*, *prefix*, *changelog=None*)

Request gerrit change

class did.plugins.gerrit.**Gerrit**(*baseurl*, *prefix*)

curl -s 'https://REPOURL/gerrit/changes/?q=is:abandoned+age:7d'

get_changelog(*chg*)

get_query_result(*url*)

static join_URL_frags(*base*, *query*)

search(*query*)

class did.plugins.gerrit.**GerritStats**(*option*, *name=None*, *parent=None*, *user=None*)

Gerrit

order = 350

class did.plugins.gerrit.**GerritUnit**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

General mother class offering general services for querying Gerrit repo.

fetch(*query_string=''*, *common_query_options=None*, *limit_since=False*)

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

static get_gerrit_date(*instr*)

class did.plugins.gerrit.**MergedChanges**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

Changes successfully merged

fetch()

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

class did.plugins.gerrit.**ReviewedChanges**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

Review of a change (for reviewers)

fetch()

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

class did.plugins.gerrit.**SubmitedChanges**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

Changes submitted for review

fetch()

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

class did.plugins.gerrit.**WIPChanges**(*option*, *name=None*, *parent=None*, *base_url=None*, *prefix=None*)

Work in progress changes

fetch()

Backend for the actual gerrit query.

query_string: basic query terms, e.g., 'status:abandoned'

common_query_options: [optional] rest of the query string; if omitted, the default one is used (limit by the current user and since option); if empty, nothing will be added to query_string

limit_since: [optional] Boolean (defaults to False) post-process the results to eliminate items created after since option.

1.5.5 git

Git commits

Config example:

```
[tools]
type = git
did = ~/git/did
edd = ~/git/edd
fmf = ~/git/fmf
```

(continues on next page)

(continued from previous page)

```
[tests]
type = git
fedora = ~/tests/fedora/*
rhel = ~/tests/rhel/*
```

Note that using an * you can enable multiple git repositories at once. Non git directories from the expansion are silently ignored.

class did.plugins.git.**GitCommits**(*option*, *name=None*, *parent=None*, *path=None*)
Git commits

fetch()
Fetch the stats (to be implemented by respective class).

header()
Show summary header.

class did.plugins.git.**GitRepo**(*path*)
Git repository investigator

commits(*user*, *options*)
List commits for given user.

class did.plugins.git.**GitStats**(*option*, *name=None*, *parent=None*, *user=None*)
Git stats group

order = 300

1.5.6 github

GitHub stats such as created and closed issues

Config example:

```
[github]
type = github
url = https://api.github.com/
token = <authentication-token>
login = <username>
```

The authentication token is optional. However, unauthenticated queries are limited. For more details see **'GitHub API'__** docs. Use `login` to override the default email address for searching. See the *Config* documentation for details on using aliases.

class did.plugins.github.**GitHub**(*url*, *token*)
GitHub Investigator

search(*query*)
Perform GitHub query

class did.plugins.github.**GitHubStats**(*option*, *name=None*, *parent=None*, *user=None*)
GitHub work

order = 330

class did.plugins.github.**Issue**(*data*)
GitHub Issue

class did.plugins.github.**IssuesClosed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Issues closed

fetch()

Fetch the stats (to be implemented by respective class).

class did.plugins.github.**IssuesCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Issues created

fetch()

Fetch the stats (to be implemented by respective class).

class did.plugins.github.**PullRequestsClosed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Pull requests closed

fetch()

Fetch the stats (to be implemented by respective class).

class did.plugins.github.**PullRequestsCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Pull requests created

fetch()

Fetch the stats (to be implemented by respective class).

class did.plugins.github.**PullRequestsReviewed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Pull requests reviewed

fetch()

Fetch the stats (to be implemented by respective class).

1.5.7 gitlab

GitLab stats such as created and closed issues

Config example:

```
[gitlab]
type = gitlab
url = https://gitlab.com/
token = <authentication-token>
login = <username>
ssl_verify = true
```

The authentication token is required. Create it in the GitLab web interface (select `api` as the desired scope). Use `login` to override user name detected from the email address. See the *Config* documentation for details on using aliases. Use `ssl_verify` to enable/disable SSL verification (default: true)

class did.plugins.gitlab.**GitLab**(*url*, *token*, *ssl_verify=True*)

GitLab Investigator

get_project(*project_id*)

get_project_issue(*project_id*, *issue_id*)

get_project_issues(*project_id*)

get_project_mr(*project_id*, *mr_id*)

get_project_mrs(*project_id*)

get_user(*username*)

search(*user*, *since*, *until*, *target_type*, *action_name*)
Perform GitLab query

user_events(*user_id*, *since*, *until*)

class did.plugins.gitlab.**GitLabStats**(*option*, *name=None*, *parent=None*, *user=None*)
GitLab work

order = 380

class did.plugins.gitlab.**Issue**(*data*, *gitlabapi*)
GitLab Issue

iid()

class did.plugins.gitlab.**IssuesClosed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issue closed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**IssuesCommented**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issue commented

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**IssuesCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issue created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**MergeRequest**(*data*, *gitlabapi*)

iid()

class did.plugins.gitlab.**MergeRequestsApproved**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Merge requests approved

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**MergeRequestsClosed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Merge requests closed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**MergeRequestsCommented**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
MergeRequests commented

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**MergeRequestsCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)

Merge requests created

fetch()

Fetch the stats (to be implemented by respective class).

class did.plugins.gitlab.**Note**(*data*, *gitlabapi*)

iid()

1.5.8 google

Google Apps stats such as attended events, completed tasks or sent emails

Config example:

```
[google]
type = google
client_id = <client_id>
client_secret = <client_secret>
apps = calendar,tasks
storage = ~/.did/google-api-credentials.json
```

Make sure you have additional dependencies of the google plugin installed on your system:

```
sudo dnf install python2-google-api-client      # Fedora
pip install did[google]                         # pip
```

To retrieve data via Google API, you will need to create access credentials (`client_id` and `client_secret`) first. Perform the following steps to create such a pair:

1. Open https://console.developers.google.com/flows/enableapi?apiid=calendar,tasks
2. In the drop-down menu, select *Create project* and click *Continue*
3. Click *Go to credentials*
4. In the *Where will you be calling the API from?* drop-down menu, choose *Other UI (e.g. Windows, CLI tool)*
5. In *What data will you be accessing?*, choose *User data*
6. Click *What credentials do I need?*
7. Input 'did credentials' in the *Name* field and click *Create client ID*
8. In *Product name shown to users*, type 'did'
9. Click *Continue*, then *Done*
10. Click the *did credentials* link to display the credentials

The `apps` configuration option defines the scope of user data the application will request (read-only) access to. Currently, the only supported values are `calendar` and `tasks`.

During the first run, user will be asked to grant the plugin access rights to selected apps. If the user approves the request, this decision is remembered by creating a *credential storage* file. The path to the storage can be customized by configuring the `storage` option.

class did.plugins.google.**Event**(*dict*)

Googe Calendar Event

attended_by(*email*)
Check if user attended the event

created_by(*email*)
Check if user created the event

organized_by(*email*)
Check if user created the event

class did.plugins.google.**GoogleCalendar**(*http*)
Google Calendar functions

events(***kwargs*)
Fetch events meeting specified criteria

class did.plugins.google.**GoogleEventsAttended**(*option*, *name=None*, *parent=None*)
Events attended

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.google.**GoogleEventsOrganized**(*option*, *name=None*, *parent=None*)
Events organized

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.google.**GoogleStatsBase**(*option*, *name=None*, *parent=None*)
Base class containing common code

events
All events in calendar within specified time range

tasks
All completed tasks within specified time range

class did.plugins.google.**GoogleStatsGroup**(*option*, *name=None*, *parent=None*, *user=None*)
Google stats group

order = 50

class did.plugins.google.**GoogleTasks**(*http*)
Google Tasks functions

tasks(***kwargs*)
Fetch tasks specified criteria

class did.plugins.google.**GoogleTasksCompleted**(*option*, *name=None*, *parent=None*)
Tasks completed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.google.**Task**(*dict*)
Google Tasks task

did.plugins.google.**authorized_http**(*client_id*, *client_secret*, *apps*, *file=None*)
Start an authorized HTTP session.

Try fetching valid user credentials from storage. If nothing has been stored, or if the stored credentials are invalid, complete the OAuth2 flow to obtain new credentials.

1.5.9 header

Customizable header

Config example:

```
[header]
type = header
highlights = Highlights
joy = Joy of the week ;-)
```

class did.plugins.header.**Header**(*option*, *name=None*, *parent=None*, *user=None*)

order = 0

1.5.10 items

Custom section with multiple items

Config example:

```
[projects]
type = items
header = Work on projects
item1 = Project One
item2 = Project Two
item3 = Project Three
```

class did.plugins.items.**CustomStats**(*option*, *name=None*, *parent=None*, *user=None*)
Custom stats

order = 800

class did.plugins.items.**ItemStats**(*option*, *name=None*, *parent=None*)
Custom section with given items

fetch()
Fetch the stats (to be implemented by respective class).

header()
Simple header for custom stats (no item count)

1.5.11 jira

Jira stats such as created, updated or resolved issues

Configuration example (GSS authentication):

```
[jboss]
type = jira
url = https://issues.jboss.org/
ssl_verify = true
```

Configuration example (basic authentication) with alternative username and custom prefix:

```
[jboss]
type = jira
prefix = JIRA
login = alt_username
url = https://issues.jboss.org/
auth_url = https://issues.jboss.org/rest/auth/latest/session
auth_type = basic
auth_username = username
auth_password = password
auth_password_file = ~/.did/jira_password
```

Configuration example limiting report only to a single project:

```
[jboss]
type = jira
project = ORG
url = https://issues.jboss.org/
ssl_verify = true
```

Notes: * If your JIRA does not have scriptrunner installed you must set

`use_scriptrunner` to false.

- You must provide `login` variable that matches username if it doesn't match email/JIRA account.
- Optional parameter `ssl_verify` can be used to enable/disable SSL verification (default: true).
- `auth_url` parameter is optional. If not provided, `url + "/step-auth-gss"` will be used for authentication.
- `auth_type` parameter is optional, default value is 'gss'.
- `auth_username`, `auth_password` and `auth_password_file` are only valid for basic authentication, `auth_password` or `auth_password_file` must be provided, `auth_password` has a higher priority.

class did.plugins.jira.**Issue**(*issue=None*, *prefix=None*)
Jira issue investigator

static search(*query*, *stats*)
Perform issue search for given stats instance

updated(*user*, *options*)
True if the issue was commented by given user

class did.plugins.jira.**JiraCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Created issues

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.jira.**JiraResolved**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Resolved issues

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.jira.**JiraStats**(*option*, *name=None*, *parent=None*, *user=None*)
Jira stats

order = 600

session
Initialize the session

class did.plugins.jira.**JiraUpdated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Updated issues

fetch()
Fetch the stats (to be implemented by respective class).

1.5.12 nitrate

Nitrate stats such as created test plans, runs, cases

Config example:

```
[nitrate]
type = nitrate
```

class did.plugins.nitrate.**AutomatedCases**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Automated cases created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.nitrate.**AutoproposedCases**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Cases proposed for automation

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.nitrate.**CopiedCases**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Test cases copied

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.nitrate.**ManualCases**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Manual cases created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.nitrate.**NitrateStats**(*option*, *name=None*, *parent=None*, *user=None*)
Nitrate stats

cases
All test cases created by the user

copies
All test case copies created by the user

order = 100

class did.plugins.nitrate.**TestPlans**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Test plans created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.nitrate.**TestRuns**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Test runs finished

fetch()
Fetch the stats (to be implemented by respective class).

1.5.13 pagure

Pagure stats such as created/closed issues and pull requests.

Config example:

```
[pagure]
type = pagure
url = https://pagure.io/api/0/
login = <username>
token = <authentication-token>
```

Use `login` to override the default email address for searching. See the *Config* documentation for details on using aliases. The authentication token is optional.

class did.plugins.pagure.**Issue**(*data*)
Pagure Issue or Pull Request

class did.plugins.pagure.**IssuesClosed**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issues closed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.pagure.**IssuesCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issues created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.pagure.**Pagure**(*url*, *token*)
Pagure Investigator

search(*query*, *pagination*, *result_field*)
Perform Pagure query

class did.plugins.pagure.**PagureStats**(*option*, *name=None*, *parent=None*, *user=None*)
Pagure work

order = 390

class did.plugins.pagure.**PullRequestsCreated**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Pull requests created

fetch()
Fetch the stats (to be implemented by respective class).

1.5.14 redmine

1.5.15 rt

Request Tracker stats such as reported and resolved tickets

Config example:

```
[rt]
type = rt
prefix = RT
url = https://tracker.org/rt/Search/Results.tsv
```

class did.plugins.rt.**ReportedTickets**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Tickets reported

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.rt.**RequestTracker**(*parent*)
Request Tracker Investigator

get(*path*)
Perform a GET request with GSSAPI authentication

search(*query*)
Perform request tracker search

class did.plugins.rt.**RequestTrackerStats**(*option*, *name=None*, *parent=None*, *user=None*)
Request Tracker

order = 500

class did.plugins.rt.**ResolvedTickets**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Tickets resolved

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.rt.**Ticket**(*record*, *parent*)
Request tracker ticket

1.5.16 sentry

Sentry stats such as commented and resolved issues.

Configuration example:

```
[sentry]
type = sentry
url = https://sentry.io/api/0/
organization = team
token = ...
```

You need to generate authentication token at the server. The only scope you need to enable is *org:read*.

class did.plugins.sentry.**Activity**(*activity*)
Sentry Activity

class did.plugins.sentry.**CommentedIssues**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issues commented

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.sentry.**Issue**(*issue*)
Sentry Issue

class did.plugins.sentry.**ResolvedIssues**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Issues resolved

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.sentry.**Sentry**(*config*, *stats*)
Sentry API

activities()
Return all activites (fetch only once)

issues(*kind*, *email*)
Filter unique issues for given activity type and email

class did.plugins.sentry.**SentryStats**(*option*, *name=None*, *parent=None*, *user=None*)
Sentry stats

order = 650

1.5.17 trac

Trac stats such as created, accepted, updated and closed tickets

Config example:

```
[trac]
type = trac
prefix = TT
url = https://some.trac.com/trac/project/rpc
```

class did.plugins.trac.**Trac**(*ticket=None*, *changelog=None*, *parent=None*, *options=None*)
Trac investigator

accepted(*user*)
True if ticket was accepted in given time frame

closed()
True if ticket was closed in given time frame

history(*user=None*)
Return relevant who-did-what logs from the ticket history

static search(*query*, *parent*, *options*)
Perform Trac search

updated(*user*)
True if the user commented the ticket in given time frame

class did.plugins.trac.**TracAccepted**(*option*, *name=None*, *parent=None*)
Accepted tickets

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trac.**TracClosed**(*option*, *name=None*, *parent=None*)
Closed tickets

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trac.**TracCommon**(*option*, *name=None*, *parent=None*)
Common Trac Stats object for saving prefix & proxy

class did.plugins.trac.**TracCreated**(*option*, *name=None*, *parent=None*)
Created tickets

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trac.**TracStats**(*option*, *name=None*, *parent=None*, *user=None*)
Trac stats group

order = 400

class did.plugins.trac.**TracUpdated**(*option*, *name=None*, *parent=None*)
Updated tickets

fetch()
Fetch the stats (to be implemented by respective class).

1.5.18 trello

Trello actions such as created, moved or closed cards

Config example (public):

```
[tools]
type = trello
user = member
```

Config example (private):

```
[tools]
type = trello
apikey = ...
token = ...
```

Optional arguments:

```
board_links = g9mdhdzg
filters = createCard, updateCard,
    updateCard:idList, updateCard:closed,
    updateCheckItemStateOnCard
```

apikey https://trello.com/app-key

token http://stackoverflow.com/questions/17178907

boards default: all

filters default: all

class did.plugins.trello.**TrelloAPI**(*stats*, *config*)
Trello API

board_links_to_ids()
Convert board links to ids

get_actions(*filters*, *since=None*, *before=None*, *limit=1000*)
Example of data structure: https://api.trello.com/1/members/ben/actions?limit=2

class did.plugins.trello.**TrelloCardsClosed**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello cards closed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloCardsCommented**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello cards commented

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloCardsCreated**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello cards created

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloCardsMoved**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello cards moved

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloCardsUpdated**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello cards updated

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloCheckItem**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello checklist items completed

fetch()
Fetch the stats (to be implemented by respective class).

class did.plugins.trello.**TrelloStats**(*trello*, *filt*, *option*, *name=None*, *parent=None*)
Trello stats

class did.plugins.trello.**TrelloStatsGroup**(*option*, *name=None*, *parent=None*, *user=None*)
Trello stats group

order = 450

session
Initialize the session

1.5.19 wiki

MoinMoin wiki stats about updated pages

Config example:

```
[wiki]
type = wiki
wiki test = http://moinmo.in/
```

The optional key 'api' can be used to change the default xmlrpc api endpoint:

```
[wiki]
type = wiki
api = ?action=xmlrpc2
wiki test = http://moinmo.in/
```

class did.plugins.wiki.**WikiChanges**(*option*, *name=None*, *parent=None*, *url=None*, *api=None*)
Wiki changes

fetch()
Fetch the stats (to be implemented by respective class).

header()
Show summary header.

merge(*other*)
Merge another stats.

class did.plugins.wiki.**WikiStats**(*option*, *name=None*, *parent=None*, *user=None*)
Wiki stats

order = 700

1.6 Modules

What did you do last week, month, year?

Comfortably gather status report data (e.g. list of committed changes) for given week, month, quarter, year or selected date range. By default all available stats for this week are reported. Detailed documentation available at http://did.readthedocs.org/.

The *stats* module contains the core of the stats gathering functionality. Some basic functionality like exceptions, config, user and date handling is placed in the *base* module. Generic utilities can be found in the *utils* module. Option parsing and other command line stuff resides in the *cli* module.

1.6.1 stats

Stats & StatsGroup, the core of the data gathering

class did.stats.**EmptyStats**(*option*, *name=None*, *parent=None*, *user=None*)
Custom stats group for header & footer

fetch()
Nothing to do for empty stats

show()
Name only for empty stats

class did.stats.**EmptyStatsGroup**(*option*, *name=None*, *parent=None*, *user=None*)
Header & Footer stats group

class did.stats.**Stats**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
General statistics

add_option(*group*)
Add option for self to the parser group object.

check()
Check the stats if enabled.

dest = None

enabled()
Check whether we're enabled (or if parent is).

fetch()
Fetch the stats (to be implemented by respective class).

header()
Show summary header.

merge(*other*)
Merge another stats.

name
Use the first line of docs string unless name set.

option = None

parent = None

show()
Display indented statistics.

stats = None

class did.stats.**StatsGroup**(*option*, *name=None*, *parent=None*, *user=None*, *options=None*)
Stats group

add_option(*parser*)
Add option group and all children options.

check()
Check all children stats.

fetch()
Stats groups do not fetch anything

merge(*other*)
Merge all children stats.

order = 500

show()
List all children stats.

class did.stats.**StatsGroupPlugin**(*name*, *bases*, *attrs*)

ignore = {'EmptyStatsGroup', 'StatsGroup', 'StatsGroupPlugin', 'UserStats'}

registry = {}

class did.stats.**UserStats**(*user=None*, *options=None*, *config=None*)
User statistics in one place

add_option(*parser*)
Add options for each stats group.

configured_plugins(*config*)
Create a StatsGroup instance for each configured plugin

1.6.2 base

Config, Date, User and Exceptions

class did.base.**Config**(*config=None*, *path=None*)
User config file

email
User email(s)

static example()
Return config example

item(*section*, *it*)
Return content of given item in selected section

parser = None

static path()
Detect config file path

plugins
Custom plugins

quarter
The first month of the quarter, 1 by default

section(*section, skip=['type', 'order']*)
Return section items, skip selected (type/order by default)

sections(*kind=None*)
Return all sections (optionally of given kind only)

width
Maximum width of the report

exception did.base.**ConfigError**
Stats configuration problem

exception did.base.**ConfigFileError**
Problem with the config file

class did.base.**Date**(*date=None*)
Date parsing for common word formats

static last_month()
Return start and end date of this month.

static last_quarter()
Return start and end date of this quarter.

static last_week()
Return start and end date of the last week.

static last_year()
Return start and end date of the last fiscal year

static period(*argument*)
Detect desired time period for the argument

static this_month()
Return start and end date of this month.

static this_quarter()
Return start and end date of this quarter.

static this_week()
Return start and end date of the current week.

static this_year()
Return start and end date of this fiscal year

exception did.base.**GeneralError**
General stats error

exception did.base.**OptionError**
Invalid command line

exception did.base.**ReportError**
Report generation error

class did.base.**User**(*email*, *stats=None*)
User information

The User object holds name, login and email which are used for performing queries by individual plugins. This information is parsed from given email address. Both short & full email format are supported:

```
some@email.org
Name Surname <some@email.org>
```

In addition, it's possible to provide email and login aliases for individual stats. This is useful if you use different email/login for different services. The syntax consists of `stats: login` or `stats: email` pairs appended at the end of the email address:

```
some@email.org; bz: bugzilla@email.org; gh: githublogin
```

Use config section name to identify stats where given alias should be used. The exactly same syntax can be used both in the config file and on the command line. Finally it's also possible to include the alias directly in the respective config section:

```
[github]
type = github
url = https://api.github.com/
login = psss
```

alias(*aliases*, *stats*)
Apply the login/email alias if configured.

clone(*stats*)
Create a user copy with alias enabled for given stats.

1.6.3 utils

Logging, config, constants & utilities

class did.utils.**Coloring**(*mode=None*)
Coloring configuration

MODES = ['COLOR_OFF', 'COLOR_ON', 'COLOR_AUTO']

enabled()
True if coloring is currently enabled

get()
Get the current color mode

set(*mode=None*)
Set the coloring mode

If enabled, some objects (like case run Status) are printed in color to easily spot failures, errors and so on. By default the feature is enabled when script is attached to a terminal. Possible values are:

```
COLOR=0 ... COLOR_OFF .... coloring disabled
COLOR=1 ... COLOR_ON ..... coloring enabled
COLOR=2 ... COLOR_AUTO ... if terminal attached (default)
```

Environment variable COLOR can be used to set up the coloring to the desired mode without modifying code.

class did.utils.**Logging**(*name='did'*)
Logging Configuration

COLORS = {4: 'magenta', 7: 'cyan', 10: 'green', 20: 'blue', 30: 'yellow', 40: 'r

class ColoredFormatter(*fmt=None*, *datefmt=None*, *style='%'*)
Custom color formatter for logging

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

LEVELS = ['CRITICAL', 'DEBUG', 'ERROR', 'FATAL', 'INFO', 'NOTSET', 'WARN', 'WARNING']

MAPPING = {0: 30, 1: 20, 2: 10, 3: 7, 4: 4, 5: 1}

get()
Get the current log level

set(*level=None*)
Set the default log level

If the level is not specified environment variable DEBUG is used with the following meaning:

```
DEBUG=0 ... LOG_WARN (default)
DEBUG=1 ... LOG_INFO
DEBUG=2 ... LOG_DEBUG
DEBUG=3 ... LOG_DETAILS
DEBUG=4 ... LOG_DATA
DEBUG=5 ... LOG_ALL (log all messages)
```

did.utils.**ascii**(*text*)
Transliterate special unicode characters into pure ascii

did.utils.**color**(*text*, *color=None*, *background=None*, *light=False*, *enabled=True*)
Return text in desired color if coloring enabled

Available colors: black red green yellow blue magenta cyan white. Alternatively color can be prefixed with "light", e.g. lightgreen.

did.utils.**header**(*text*)
Show text as a header.

did.utils.**info**(*message*, *newline=True*)
Log provided info message to the standard error output

did.utils.**item**(*text*, *level=0*, *options=None*)
Print indented item.

did.utils.**listed**(*items*, *singular=None*, *plural=None*, *max=None*, *quote=''*)
Convert an iterable into a nice, human readable list or description:

```
listed(range(1)) ..................... 0
listed(range(2)) ..................... 0 and 1
listed(range(3), quote='"') ......... "0", "1" and "2"
listed(range(4), max=3) .............. 0, 1, 2 and 1 more
listed(range(5), 'number', max=3) ... 0, 1, 2 and 2 more numbers
listed(range(6), 'category') ........ 6 categories
listed(7, "leaf", "leaves") ......... 7 leaves
```

If singular form is provided but max not set the description-only mode is activated as shown in the last two examples. Also, an int can be used in this case to get a simple inflection functionality.

did.utils.**load_components**(**paths*, ***kwargs*)
Load all components on the paths

Each path should be a package or module. All components beneath a path are loaded. This method works whether the package or module is on the filesystem or in an .egg. If it's in an egg, the egg must already be on the PYTHONPATH.

Args: paths (str): A package or module to load

Keyword Args:

include (str): A regular expression of packages and modules to include. Defaults to '.*'

exclude (str): A regular expression of packges and modules to exclude. Defaults to 'test'

continue_on_error (bool): If True, continue importing even if something raises an ImportError. If False, raise the first ImportError.

Returns: int: The total number of modules loaded.

Raises: ImportError

did.utils.**pluralize**(*singular=None*)
Naively pluralize words

did.utils.**shorted**(*text*, *width=79*)
Shorten text, make sure it's not cut in the middle of a word

did.utils.**split**(*values*, *separator=re.compile('[ ], +')*)
Convert space-or-comma-separated values into a single list

Common use case for this is merging content of options with multiple values allowed into a single list of strings thus allowing any of the formats below and converts them into ['a', 'b', 'c']:

```
--option a --option b --option c ... ['a', 'b', 'c']
--option a,b --option c ............ ['a,b', 'c']
--option 'a b c' ................... ['a b c']
```

Accepts both string and list. By default space and comma are used as value separators. Use any regular expression for custom separator.

1.6.4 cli

Command line interface for did

This module takes care of processing command line options and running the main loop which gathers all individual stats.

class did.cli.**Options**(*arguments=None*)
Command line options parser

check()
Perform additional check for given options

parse()
Parse the options.

did.cli.**main**(*arguments=None*)
Parse options, gather stats and show the results

Takes optional parameter `arguments` which can be either command line string or list of options. This is very useful for testing purposes. Function returns a tuple of the form:

```
([user_stats], team_stats)
```

with the list of all gathered stats objects.

1.7 Contribute

1.7.1 Introduction

Feel free and welcome to contribute to this project. You can start with filing issues and ideas for improvement in GitHub tracker. My favorite thoughts from The Zen of Python:

- Beautiful is better than ugly.
- Simple is better than complex.
- Readability counts.

A couple of recommendations from PEP8 and myself:

- Comments should be complete sentences.
- The first word should be capitalized (unless identifier).
- When using hanging indent, the first line should be empty.

1.7.2 Start

Clone the git repository, create a virtual environment and use `pip` to install `did` with all necessary dependencies. This will also link python modules from the repo so that you can test directly the modified code:

```
git clone https://github.com/psss/did
cd did
mkvirtualenv did
pip install --editable .[all]
```

Install virtualenvwrapper if you do not have it already on your system. It makes working with virtual environments much easier.

1.7.3 Makefile

There are several Makefile targets defined to make the common daily tasks easy & efficient:

make test Execute the test suite.

make smoke Perform quick basic functionality test.

make coverage Run the test suite under coverage and report results.

make docs Build documentation.

make packages Build rpm and srpm packages.

make hooks Link git commit hooks.

make tags Create or update the Vim `tags` file for quick searching. You might want to use `set tags=./tags;` in your `.vimrc` to enable parent directory search for the tags file as well.

make clean Cleanup all temporary files.

1.7.4 Commits

It is challenging to be both concise and descriptive, but that is what a well-written summary should do. Consider the commit message as something that could/will be pasted into release notes:

- The first line should have up to 50 characters.
- Complete sentence with the first word capitalized.
- No prefixes such as "doc:" or "<filename>:".
- Should concisely describe the purpose of the patch.
- Other details should be separated by a blank line.

Why should I care?

- It helps others (and yourself) find relevant commits quickly.
- The summary line can be re-used later (e.g. for rpm changelog).
- Some tools do not handle wrapping, so it is then hard to read.
- You will make the maintainer happy to read beautiful commits :)

You can get some more context in the stackoverflow article.

1.7.5 Hooks

You can find git commit hooks in the `examples` directory. Consider linking or copying them into your git config:

```
GIT=~/git/did # Update to your actual path
ln -snf $GIT/hooks/pre-commit $GIT/.git/hooks
ln -snf $GIT/hooks/commit-msg $GIT/.git/hooks
```

Or simply run `make hooks` which will do the linking for you. Note that this will overwrite existing hooks.

1.7.6 Tests

To run tests using pytest:

```
coverage run --source=did -m py.test tests
coverage report
```

Install pytest and coverage using yum:

```
yum install pytest python-coverage
```

or pip:

```
pip install .[tests]
```

See Travis CI and Coveralls for the latest test/coverage results:

- https://travis-ci.org/psss/did/builds
- https://coveralls.io/github/psss/did

1.7.7 Docs

For building documentation locally install necessary modules:

```
pip install .[docs]
```

Building documentation is then quite straightforward:

```
make docs
```

Find the resulting html pages under the `docs/_build/html` folder.

1.7.8 MrBob

You can use also use *mrbob* to easily create templates to help you get started contributing:

```
pip install mr.bob
mrbob examples/mr.bob/plugin -O ./did/plugins
```

mrbob should have asked you a few questions before creating a new basic Stats plugin for you in *did/plugins/*. Check *git status* to see the new files it created as a result.

1.8 Questions

1.8.1 Certificates

I'm getting an *SSLError* when accessing stats.

The default certificate setup on your system should work for most cases. If you have additional certificates installed or have them stored in a different location exporting the following environment variable might help you:

```
export REQUESTS_CA_BUNDLE=/etc/pki/tls/certs/ca-bundle.crt
```

CHAPTER 2

Indices and Tables

- genindex
- modindex

Python Module Index

d

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W