Note: this page documents the version 1.0 of Trac, see 0.12/TracStandalone? if you need the previous version

Tracd

Tracd is a lightweight standalone Trac web server. It can be used in a variety of situations, from a test or development server to a multiprocess setup behind another web server used as a load balancer.

Pros

- Fewer dependencies: You don't need to install apache or any other web-server.
- Fast: Should be almost as fast as the mod_python version (and much faster than the CGI), even more so since version 0.12 where the HTTP/1.1 version of the protocol is enabled by default
- Automatic reloading: For development, Tracd can be used in *auto_reload* mode, which will automatically restart the server whenever you make a change to the code (in Trac itself or in a plugin).

Cons

- Fewer features: Tracd implements a very simple web-server and is not as configurable or as scalable as Apache httpd.
- No native HTTPS support: ?sslwrap can be used instead, or ?stunnel -- a tutorial on how to use stunnel with tracd or Apache with mod_proxy.

Usage examples

A single project on port 8080. (?http://localhost:8080/)

```
$ tracd -p 8080 /path/to/project
```

Stricly speaking this will make your Trac accessible to everybody from your network rather than *localhost only*. To truly limit it use *--hostname* option.

```
$ tracd --hostname=localhost -p 8080 /path/to/project
```

With more than one project. (?http://localhost:8080/project1/ and ?http://localhost:8080/project2/)

```
$ tracd -p 8080 /path/to/project1 /path/to/project2
```

You can't have the last portion of the path identical between the projects since Trac uses that name to keep the URLs of the different projects unique. So if you use `/project1/path/to` and `/project2/path/to`, you will only see the second project.

An alternative way to serve multiple projects is to specify a parent directory in which each subdirectory is a Trac project, using the `-e` option. The example above could be rewritten:

```
$ tracd -p 8080 -e /path/to
```

To exit the server on Windows, be sure to use `CTRL-BREAK` -- using `CTRL-C` will leave a Python process running in the background.

Installing as a Windows Service

Option 1

To install as a Windows service, get the ?SRVANY utility and run:

```
C:\path\to\instsrv.exe tracd C:\path\to\srvany.exe
reg add HKLM\SYSTEM\CurrentControlSet\Services\tracd\Parameters /v Application /d "\"C:\path\to\python
net start tracd
```

DO NOT use `tracd.exe`. Instead register `python.exe` directly with `tracd-script.py` as a parameter. If you use `tracd.exe`, it will spawn the python process without SRVANY's knowledge. This python process will survive a `net stop tracd`.

If you want tracd to start automatically when you boot Windows, do:

```
sc config tracd start= auto
```

The spacing here is important.

Once the service is installed, it might be simpler to run the Registry Editor rather than use the `reg add` command documented above. Navigate to:
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\tracd\Parameters`

Three (string) parameters are provided:

AppDirectory	C:\Python26\
Application	python.exe
AppParameters	scripts\tracd-script.py -p 8080 ...

Note that, if the AppDirectory is set as above, the paths of the executable *and* of the script name and parameter values are relative to the directory. This makes updating Python a little simpler because the change can be limited, here, to a single point. (This is true for the path to the .htpasswd file, as well, despite the documentation calling out the /full/path/to/htpasswd; however, you may not wish to store that file under the Python directory.)

For Windows 7 User, srvany.exe may not be an option, so you can use ?WINSERV utility and run:

```
"C:\path\to\winserv.exe" install tracd -displayname "tracd" -start auto "C:\path\to\python.exe" c:\path

net start tracd
```

Option 2

Use ?WindowsServiceScript, available at ?Trac Hacks. Installs, removes, starts, stops, etc. your Trac service.

Option 3

also cygwin's cygrunsrv.exe can be used:

```
$ cygrunsrv --install tracd --path /cygdrive/c/Python27/Scripts/tracd.exe --args '--port 8000 --env-par
$ net start tracd
```

Using Authentication

Tracd provides support for both Basic and Digest authentication. Digest is considered more secure. The examples below use Digest; to use Basic authentication, replace `--auth` with `--basic-auth` in the command line.

The general format for using authentication is:

```
$ tracd -p port --auth="base_project_dir,password_file_path,realm" project_path
```

where:

- **base_project_dir**: the base directory of the project specified as follows:
  - when serving multiple projects: *relative* to the `project_path`
  - when serving only a single project (`-s`): the name of the project directory

  Don't use an absolute path here as this won't work. *Note:* This parameter is case-sensitive even for environments on Windows.

- **password_file_path**: path to the password file
- **realm**: the realm name (can be anything)
- **project_path**: path of the project

- **`--auth`** in the above means use Digest authentication, replace `--auth` with `--basic-auth` if you want to use Basic auth. Although Basic authentication does not require a "realm", the command parser does, so the second comma is required, followed directly by the closing quote for an empty realm name.

Examples:

```
$ tracd -p 8080 \
  --auth="project1,/path/to/passwordfile,mycompany.com" /path/to/project1
```

Of course, the password file can be be shared so that it is used for more than one project:

```
$ tracd -p 8080 \
  --auth="project1,/path/to/passwordfile,mycompany.com" \
  --auth="project2,/path/to/passwordfile,mycompany.com" \
  /path/to/project1 /path/to/project2
```

Another way to share the password file is to specify "*" for the project name:

```
$ tracd -p 8080 \
  --auth="*,/path/to/users.htdigest,mycompany.com" \
  /path/to/project1 /path/to/project2
```

Basic Authorization: Using a htpasswd password file

This section describes how to use `tracd` with Apache .htpasswd files.

> Note: It is necessary (at least with Python 2.6) to install the fcrypt package in order to decode some htpasswd formats. Trac source code attempt an `import crypt` first, but there is no such package for Python 2.6. Only `SHA-1` passwords (since Trac 1.0) work without this module.

To create a .htpasswd file use Apache's `htpasswd` command (see below for a method to create these files without using Apache):

```
$ sudo htpasswd -c /path/to/env/.htpasswd username
```

then for additional users:

```
$ sudo htpasswd /path/to/env/.htpasswd username2
```

Then to start `tracd` run something like this:

```
$ tracd -p 8080 --basic-auth="projectdirname,/fullpath/environmentname/.htpasswd,realmname" /fullpath/
```

For example:

```
$ tracd -p 8080 --basic-auth="testenv,/srv/tracenv/testenv/.htpasswd,My Test Env" /srv/tracenv/testenv
```

Note: You might need to pass "-m" as a parameter to htpasswd on some platforms (OpenBSD).

Digest authentication: Using a htdigest password file

If you have Apache available, you can use the htdigest command to generate the password file. Type 'htdigest' to get some usage instructions, or read ?this page from the Apache manual to get precise instructions. You'll be prompted for a password to enter for each user that you create. For the name of the password file, you can use whatever you like, but if you use something like `users.htdigest` it will remind you what the file contains. As a suggestion, put it in your <projectname>/conf folder along with the trac.ini file.

Note that you can start tracd without the `--auth` argument, but if you click on the *Login* link you will get an error.

Generating Passwords Without Apache

Basic Authorization can be accomplished via this ?online HTTP Password generator which also supports `SHA-1`. Copy the generated password-hash line to the .htpasswd file on your system. Note that Windows Python lacks the "crypt" module that is the default hash type for htpasswd ; Windows Python can grok MD5 password hashes just fine and you should use MD5.

You can use this simple Python script to generate a **digest** password file:

```
from optparse import OptionParser
# The md5 module is deprecated in Python 2.5
try:
    from hashlib import md5
except ImportError:
    from md5 import md5
realm = 'trac'

# build the options
usage = "usage: %prog [options]"
parser = OptionParser(usage=usage)
parser.add_option("-u", "--username",action="store", dest="username", type = "string",
                  help="the username for whom to generate a password")
parser.add_option("-p", "--password",action="store", dest="password", type = "string",
                  help="the password to use")
parser.add_option("-r", "--realm",action="store", dest="realm", type = "string",
                  help="the realm in which to create the digest")
(options, args) = parser.parse_args()

# check options
if (options.username is None) or (options.password is None):
   parser.error("You must supply both the username and password")
if (options.realm is not None):
```

```
    realm = options.realm

# Generate the string to enter into the htdigest file
kd = lambda x: md5(':'.join(x)).hexdigest()
print ':'.join((options.username, realm, kd([options.username, realm, options.password])))
```

Note: If you use the above script you must set the realm in the `--auth` argument to **`trac`**. Example usage (assuming you saved the script as trac-digest.py):

```
 $ python trac-digest.py -u username -p password >> c:\digest.txt
 $ tracd --port 8000 --auth=proj_name,c:\digest.txt,trac c:\path\to\proj_name
```

Using `md5sum`

It is possible to use `md5sum` utility to generate digest-password file:

```
user=
realm=
password=
path_to_file=
echo ${user}:${realm}:$(printf "${user}:${realm}:${password}" | md5sum - | sed -e 's/\s\+-//') > ${path
```

Reference

Here's the online help, as a reminder (`tracd --help`):

```
Usage: tracd [options] [projenv] ...

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -a DIGESTAUTH, --auth=DIGESTAUTH
                        [projectdir],[htdigest_file],[realm]
  --basic-auth=BASICAUTH
                        [projectdir],[htpasswd_file],[realm]
  -p PORT, --port=PORT  the port number to bind to
  -b HOSTNAME, --hostname=HOSTNAME
                        the host name or IP address to bind to
  --protocol=PROTOCOL   http|scgi|ajp|fcgi
  -q, --unquote         unquote PATH_INFO (may be needed when using ajp)
  --http10              use HTTP/1.0 protocol version instead of HTTP/1.1
  --http11              use HTTP/1.1 protocol version (default)
  -e PARENTDIR, --env-parent-dir=PARENTDIR
                        parent directory of the project environments
  --base-path=BASE_PATH
                        the initial portion of the request URL's "path"
  -r, --auto-reload     restart automatically when sources are modified
  -s, --single-env      only serve a single project without the project list
  -d, --daemonize       run in the background as a daemon
  --pidfile=PIDFILE     when daemonizing, file to which to write pid
  --umask=MASK          when daemonizing, file mode creation mask to use, in
                        octal notation (default 022)
  --group=GROUP         the group to run as
  --user=USER           the user to run as
```

Use the -d option so that tracd doesn't hang if you close the terminal window where tracd was started.

Tips

Serving static content

If `tracd` is the only web server used for the project, it can also be used to distribute static content (tarballs, Doxygen documentation, etc.)

This static content should be put in the `$TRAC_ENV/htdocs` folder, and is accessed by URLs like `<project_URL>/chrome/site/...`.

Example: given a `$TRAC_ENV/htdocs/software-0.1.tar.gz` file, the corresponding relative URL would be `/<project_name>/chrome/site/software-0.1.tar.gz`, which in turn can be written as `htdocs:software-0.1.tar.gz` (TracLinks syntax) or `[/<project_name>/chrome/site/software-0.1.tar.gz]` (relative link syntax).

> *Support for `htdocs`: TracLinks syntax was added in version 0.10*

Using tracd behind a proxy

In some situations when you choose to use tracd behind Apache or another web server.

In this situation, you might experience issues with redirects, like being redirected to URLs with the wrong host or protocol. In this case (and only in this case), setting the `[trac] use_base_url_for_redirect` to `true` can help, as this will force Trac to use the value of `[trac] base_url` for doing the redirects.

If you're using the AJP protocol to connect with `tracd` (which is possible if you have flup installed), then you might experience problems with double quoting. Consider adding the `--unquote` parameter.

See also ?TracOnWindowsIisAjp, ?TracNginxRecipe.

Authentication for tracd behind a proxy

It is convenient to provide central external authentication to your tracd instances, instead of using `--basic-auth`. There is some discussion about this in #9206.

Below is example configuration based on Apache 2.2, mod_proxy, mod_authnz_ldap.

First we bring tracd into Apache's location namespace.

```
<Location /project/proxified>
        Require ldap-group cn=somegroup, ou=Groups,dc=domain.com
        Require ldap-user somespecificusertoo
        ProxyPass http://localhost:8101/project/proxified/
        # Turns out we don't really need complicated RewriteRules here at all
        RequestHeader set REMOTE_USER %{REMOTE_USER}s
</Location>
```

Then we need a single file plugin to recognize HTTP_REMOTE_USER header as valid authentication source. HTTP headers like **HTTP_FOO_BAR** will get converted to **Foo-Bar** during processing. Name it something like **remote-user-auth.py** and drop it into **proxified/plugins** directory:

```
from trac.core import *
from trac.config import BoolOption
from trac.web.api import IAuthenticator
```

```
class MyRemoteUserAuthenticator(Component):

    implements(IAuthenticator)

    obey_remote_user_header = BoolOption('trac', 'obey_remote_user_header', 'false',
               """Whether the 'Remote-User:' HTTP header is to be trusted for user logins
                (''since ??.??').""")

    def authenticate(self, req):
        if self.obey_remote_user_header and req.get_header('Remote-User'):
            return req.get_header('Remote-User')
        return None
```

Add this new parameter to your TracIni:

```
...
[trac]
...
obey_remote_user_header = true
...
```

Run tracd:

```
tracd -p 8101 -r -s proxified --base-path=/project/proxified
```

Note that if you want to install this plugin for all projects, you have to put it in your global plugins_dir and enable it in your global trac.ini.

Global config (e.g. `/srv/trac/conf/trac.ini`):

```
[components]
remote-user-auth.* = enabled
[inherit]
plugins_dir = /srv/trac/plugins
[trac]
obey_remote_user_header = true
```

Environment config (e.g. `/srv/trac/envs/myenv`):

```
[inherit]
file = /srv/trac/conf/trac.ini
```

Serving a different base path than /

Tracd supports serving projects with different base urls than /<project>. The parameter name to change this is

```
 $ tracd --base-path=/some/path
```

See also: TracInstall, TracCgi, TracModPython, TracGuide, ?Running tracd.exe as a Windows service