Certbot Documentation

Release 0.10.0.dev0

Certbot Project

Jan 25, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

Certbot is part of EFF's effort to encrypt the entire Internet. Secure communication over the Web relies on HTTPS, which requires the use of a digital certificate that lets browsers verify the identify of web servers (e.g., is that really google.com?). Web servers obtain their certificates from trusted third parties called certificate authorities (CAs). Certbot is an easy-to-use client that fetches a certificate from Let's Encrypt—an open certificate authority launched by the EFF, Mozilla, and others—and deploys it to a web server.

Anyone who has gone through the trouble of setting up a secure website knows what a hassle getting and maintaining a certificate is. Certbot and Let's Encrypt can automate away the pain and let you turn on and manage HTTPS with simple commands. Using Certbot and Let's Encrypt is free, so there's no need to arrange payment.

How you use Certbot depends on the configuration of your web server. The best way to get started is to use our interactive guide. It generates instructions based on your configuration settings. In most cases, you'll need root or administrator access to your web server to run Certbot.

If you're using a hosted service and don't have direct access to your web server, you might not be able to use Certbot. Check with your hosting provider for documentation about uploading certificates or using certificates issues by Let's Encrypt.

Certbot is a fully-featured, extensible client for the Let's Encrypt CA (or any other CA that speaks the ACME protocol) that can automate the tasks of obtaining certificates and configuring webservers to use them. This client runs on Unix-based operating systems.

Until May 2016, Certbot was named simply `letsencrypt` or `letsencrypt-auto`, depending on install method. Instructions on the Internet, and some pieces of the software, may still refer to this older name.

Contributing

If you'd like to contribute to this project please read Developer Guide.

Installation

The easiest way to install Certbot is by visiting certbot.eff.org, where you can find the correct installation instructions for many web server and OS combinations. For more information, see Get Certbot.

How to run the client

In many cases, you can just run `certbot-auto` or `certbot`, and the client will guide you through the process of obtaining and installing certs interactively.

For full command line help, you can type:

```
./certbot-auto --help all
```

You can also tell it exactly what you want it to do from the command line. For instance, if you want to obtain a cert for `example.com`, `www.example.com`, and `other.example.net`, using the Apache plugin to both obtain and install the certs, you could do this:

```
./certbot-auto --apache -d example.com -d www.example.com -d other.example.net
```

(The first time you run the command, it will make an account, and ask for an email and agreement to the Let's Encrypt Subscriber Agreement; you can automate those with `--email` and `--agree-tos`)

If you want to use a webserver that doesn't have full plugin support yet, you can still use "standalone" or "webroot" plugins to obtain a certificate:

```
./certbot-auto certonly --standalone --email admin@example.com -d example.com -d www.
↪example.com -d other.example.net
```

Understanding the client in more depth

To understand what the client is doing in detail, it's important to understand the way it uses plugins. Please see the explanation of plugins in the User Guide.

Links

Documentation: https://certbot.eff.org/docs

Software project: https://github.com/certbot/certbot

Notes for developers: https://certbot.eff.org/docs/contributing.html

Main Website: https://certbot.eff.org

Let's Encrypt Website: https://letsencrypt.org

IRC Channel: #letsencrypt on Freenode or #certbot on OFTC

Community: https://community.letsencrypt.org

ACME spec: http://ietf-wg-acme.github.io/acme/

ACME working area in github: https://github.com/ietf-wg-acme/acme

Mailing list: client-dev (to subscribe without a Google account, send an email to client-dev+subscribe@letsencrypt.org)

System Requirements

The Let's Encrypt Client presently only runs on Unix-ish OSes that include Python 2.6 or 2.7; Python 3.x support will hopefully be added in the future. The client requires root access in order to write to `/etc/letsencrypt`, `/var/log/letsencrypt`, `/var/lib/letsencrypt`; to bind to ports 80 and 443 (if you use the `standalone` plugin) and to read and modify webserver configurations (if you use the `apache` or `nginx` plugins). If none of these apply to you, it is theoretically possible to run without root privileges, but for most users who want to avoid running an ACME client as root, either letsencrypt-nosudo or simp_le are more appropriate choices.

The Apache plugin currently requires a Debian-based OS with augeas version 1.0; this includes Ubuntu 12.04+ and Debian 7+.

CHAPTER

TWO

GET CERTBOT

Table of Contents

About Certbot

Certbot is packaged for many common operating systems and web servers. Check whether `certbot` (or `letsencrypt`) is packaged for your web server's OS by visiting certbot.eff.org, where you will also find the correct installation instructions for your system.

Note: Unless you have very specific requirements, we kindly suggest that you use the Certbot packages provided by your package manager (see certbot.eff.org). If such packages are not available, we recommend using `certbot-auto`, which automates the process of installing Certbot on your system.

System Requirements

The Let's Encrypt Client presently only runs on Unix-ish OSes that include Python 2.6 or 2.7; Python 3.x support will hopefully be added in the future. The client requires root access in order to write to `/etc/letsencrypt`, `/var/log/letsencrypt`, `/var/lib/letsencrypt`; to bind to ports 80 and 443 (if you use the `standalone` plugin) and to read and modify webserver configurations (if you use the `apache` or `nginx` plugins). If none of these apply to you, it is theoretically possible to run without root privileges, but for most users who want to avoid running an ACME client as root, either letsencrypt-nosudo or simp_le are more appropriate choices.

The Apache plugin currently requires OS with augeas version 1.0; currently it supports modern OSes based on Debian, Fedora, SUSE, Gentoo and Darwin.

Alternate installation methods

If you are offline or your operating system doesn't provide a package, you can use an alternate method for installing `certbot`.

Certbot-Auto

The `certbot-auto` wrapper script installs Certbot, obtaining some dependencies from your web server OS and putting others in a python virtual environment. You can download and run it as follows:

```
user@webserver:~$ wget https://dl.eff.org/certbot-auto
user@webserver:~$ chmod a+x ./certbot-auto
user@webserver:~$ ./certbot-auto --help
```

Hint: The certbot-auto download is protected by HTTPS, which is pretty good, but if you'd like to double check the integrity of the `certbot-auto` script, you can use these steps for verification before running it:

```
user@server:~$ wget -N https://dl.eff.org/certbot-auto.asc
user@server:~$ gpg2 --recv-key A2CFB51FA275A7286234E7B24D17C995CD9775F2
user@server:~$ gpg2 --trusted-key 4D17C995CD9775F2 --verify certbot-auto.asc certbot-
↪auto
```

The `certbot-auto` command updates to the latest client release automatically. Since `certbot-auto` is a wrapper to `certbot`, it accepts exactly the same command line flags and arguments. For more information, see Certbot command-line options.

For full command line help, you can type:

```
./certbot-auto --help all
```

Running with Docker

Docker is an amazingly simple and quick way to obtain a certificate. However, this mode of operation is unable to install certificates or configure your webserver, because our installer plugins cannot reach your webserver from inside the Docker container.

Most users should use the operating system packages (see instructions at certbot.eff.org) or, as a fallback, `certbot-auto`. You should only use Docker if you are sure you know what you are doing and have a good reason to do so.

You should definitely read the *Where are my certificates?* section, in order to know how to manage the certs manually. Our ciphersuites page provides some information about recommended ciphersuites. If none of these make much sense to you, you should definitely use the *certbot-auto* method, which enables you to use installer plugins that cover both of those hard topics.

If you're still not convinced and have decided to use this method, from the server that the domain you're requesting a cert for resolves to, install Docker, then issue the following command:

```
sudo docker run -it --rm -p 443:443 -p 80:80 --name certbot \
            -v "/etc/letsencrypt:/etc/letsencrypt" \
            -v "/var/lib/letsencrypt:/var/lib/letsencrypt" \
            quay.io/letsencrypt/letsencrypt:latest certonly
```

Running Certbot with the `certonly` command will obtain a certificate and place it in the directory `/etc/letsencrypt/live` on your system. Because Certonly cannot install the certificate from within Docker, you must install the certificate manually according to the procedure recommended by the provider of your webserver.

For more information about the layout of the `/etc/letsencrypt` directory, see *Where are my certificates?*.

Operating System Packages

FreeBSD

- Port: `cd /usr/ports/security/py-certbot && make install clean`
- Package: `pkg install py27-certbot`

OpenBSD

- Port: `cd /usr/ports/security/letsencrypt/client && make install clean`
- Package: `pkg_add letsencrypt`

Arch Linux

```
sudo pacman -S certbot
```

Debian

If you run Debian Stretch or Debian Sid, you can install certbot packages.

```
sudo apt-get update
sudo apt-get install certbot python-certbot-apache
```

If you don't want to use the Apache plugin, you can omit the `python-certbot-apache` package.

Packages exist for Debian Jessie via backports. First you'll have to follow the instructions at http://backports.debian.org/Instructions/ to enable the Jessie backports repo, if you have not already done so. Then run:

```
sudo apt-get install letsencrypt python-letsencrypt-apache -t jessie-backports
```

Fedora

```
sudo dnf install letsencrypt
```

Gentoo

The official Certbot client is available in Gentoo Portage. If you want to use the Apache plugin, it has to be installed separately:

```
emerge -av app-crypt/letsencrypt
emerge -av app-crypt/letsencrypt-apache
```

When using the Apache plugin, you will run into a "cannot find a cert or key directive" error if you're sporting the default Gentoo `httpd.conf`. You can fix this by commenting out two lines in `/etc/apache2/httpd.conf` as follows:

Change

```
<IfDefine SSL>
LoadModule ssl_module modules/mod_ssl.so
</IfDefine>
```

to

```
#<IfDefine SSL>
LoadModule ssl_module modules/mod_ssl.so
#</IfDefine>
```

For the time being, this is the only way for the Apache plugin to recognise the appropriate directives when installing the certificate. Note: this change is not required for the other plugins.

Other Operating Systems

OS packaging is an ongoing effort. If you'd like to package Certbot for your distribution of choice please have a look at the *Packaging Guide*.

Installing from source

Installation from source is only supported for developers and the whole process is described in the *Developer Guide*.

Warning: Please do **not** use `python setup.py install` or `python pip install .`. Please do **not** attempt the installation commands as superuser/root and/or without virtual environment, e.g. `sudo python setup.py install`, `sudo pip install`, `sudo ./venv/bin/...`. These modes of operation might corrupt your operating system and are **not supported** by the Certbot team!

CHAPTER

THREE

USER GUIDE

Table of Contents

Certbot Commands

Certbot uses a number of different "commands" (also referred to, equivalently, as "subcommands") to request specific actions such as obtaining, renewing, or revoking certificates. Some of the most important and most commonly-used commands will be discussed throughout this document; an exhaustive list also appears near the end of the document.

The `certbot` script on your web server might be named `letsencrypt` if your system uses an older package, or `certbot-auto` if you used an alternate installation method. Throughout the docs, whenever you see `certbot`, swap in the correct name as needed.

Getting certificates (and choosing plugins)

The Certbot client supports a number of different "plugins" that can be used to obtain and/or install certificates.

Plugins that can obtain a cert are called "authenticators" and can be used with the "certonly" command. This will carry out the steps needed to validate that you control the domain(s) you are requesting a cert for, obtain a cert for the specified domain(s), and place it in the `/etc/letsencrypt` directory on your machine - without editing any of your server's configuration files to serve the obtained certificate. If you specify multiple domains to authenticate, they will all be listed in a single certificate. To obtain multiple seperate certificates you will need to run Certbot multiple times.

Plugins that can install a cert are called "installers" and can be used with the "install" command. These plugins can modify your webserver's configuration to serve your website over HTTPS using certificates obtained by certbot.

Plugins that do both can be used with the "certbot run" command, which is the default when no command is specified. The "run" subcommand can also be used to specify a combination of distinct authenticator and installer plugins.

Plugin	Auth	Inst	Notes	Challenge types (and port)
apache	Y	Y	Automates obtaining and installing a cert with Apache 2.4 on Debian-based distributions with `libaugeas0` 1.0+.	tls-sni-01 (443)
webroot	Y	N	Obtains a cert by writing to the webroot directory of an already running webserver.	http-01 (80)
nginx	Y	Y	Automates obtaining and installing a cert with Nginx. Alpha release shipped with Certbot 0.9.0.	tls-sni-01 (443)
standalone	Y	N	Uses a "standalone" webserver to obtain a cert. Requires port 80 or 443 to be available. This is useful on systems with no webserver, or when direct integration with the local webserver is not supported or not desired.	http-01 (80) or tls-sni-01 (443)
manual	Y	N	Helps you obtain a cert by giving you instructions to perform domain validation yourself.	http-01 (80) or dns-01 (53)

Under the hood, plugins use one of several ACME protocol "Challenges" to prove you control a domain. The options are http-01 (which uses port 80), tls-sni-01 (port 443) and dns-01 (requring configuration of a DNS server on port 53, thought that's often not the same machine as your webserver). A few plugins support more than one challenge type, in which case you can choose one with `--preferred-challenges`.

There are also many *third-party-plugins* available. Below we describe in more detail the circumstances in which each plugin can be used, and how to use it.

Apache

The Apache plugin currently requires OS with augeas version 1.0; currently it supports modern OSes based on Debian, Fedora, SUSE, Gentoo and Darwin. This automates both obtaining *and* installing certs on an Apache webserver. To specify this plugin on the command line, simply include `--apache`.

Webroot

If you're running a local webserver for which you have the ability to modify the content being served, and you'd prefer not to stop the webserver during the certificate issuance process, you can use the webroot plugin to obtain a cert by including `certonly` and `--webroot` on the command line. In addition, you'll need to specify `--webroot-path` or `-w` with the top-level directory ("web root") containing the files served by your webserver. For example, `--webroot-path /var/www/html` or `--webroot-path /usr/share/nginx/html` are two common webroot paths.

If you're getting a certificate for many domains at once, the plugin needs to know where each domain's files are served from, which could potentially be a separate directory for each domain. When requesting a certificate for multiple domains, each domain will use the most recently specified `--webroot-path`. So, for instance,

```
certbot certonly --webroot -w /var/www/example/ -d www.example.com -d example.com -w /
↪var/www/other -d other.example.net -d another.other.example.net
```

would obtain a single certificate for all of those names, using the `/var/www/example` webroot directory for the first two, and `/var/www/other` for the second two.

The webroot plugin works by creating a temporary file for each of your requested domains in `${webroot-path}/.well-known/acme-challenge`. Then the Let's Encrypt validation server makes HTTP requests to validate that the DNS for each requested domain resolves to the server running certbot. An example request made to your web server would look like:

```
66.133.109.36 - - [05/Jan/2016:20:11:24 -0500] "GET /.well-known/acme-challenge/
↪HGr8U1IeTW4kY_Z6UIyaakzOkyQgPr_7ArlLgtZE8SX HTTP/1.1" 200 87 "-" "Mozilla/5.0␣
↪(compatible; Let's Encrypt validation server; +https://www.letsencrypt.org)"
```

Note that to use the webroot plugin, your server must be configured to serve files from hidden directories. If `/.well-known` is treated specially by your webserver configuration, you might need to modify the configuration to ensure that files inside `/.well-known/acme-challenge` are served by the webserver.

Nginx

The Nginx plugin has been distributed with Cerbot since version 0.9.0 and should work for most configurations. Because it is alpha code, we recommend backing up Nginx configurations before using it (though you can also revert changes to configurations with `certbot --nginx rollback`). You can use it by providing the `--nginx` flag on the commandline.

```
certbot --nginx
```

Standalone

To obtain a cert using a "standalone" webserver, you can use the standalone plugin by including `certonly` and `--standalone` on the command line. This plugin needs to bind to port 80 or 443 in order to perform domain validation, so you may need to stop your existing webserver. To control which port the plugin uses, include one of the options shown below on the command line.

- `--standalone-supported-challenges http-01` to use port 80
- `--standalone-supported-challenges tls-sni-01` to use port 443

The standalone plugin does not rely on any other server software running on the machine where you obtain the certificate. It must still be possible for that machine to accept inbound connections from the Internet on the specified port using each requested domain name.

Manual

If you'd like to obtain a cert running `certbot` on a machine other than your target webserver or perform the steps for domain validation yourself, you can use the manual plugin. While hidden from the UI, you can use the plugin to obtain a cert by specifying `certonly` and `--manual` on the command line. This requires you to copy and paste commands into another terminal session, which may be on a different computer.

Third-party plugins

There are also a number of third-party plugins for the client, provided by other developers. Many are beta/experimental, but some are already in widespread use:

Plugin	Auth	Inst	Notes
plesk	Y	Y	Integration with the Plesk web hosting tool
haproxy	Y	Y	Integration with the HAProxy load balancer
s3front	Y	Y	Integration with Amazon CloudFront distribution of S3 buckets
gandi	Y	Y	Integration with Gandi's hosting products and API
varnish	Y	N	Obtain certs via a Varnish server
external	Y	N	A plugin for convenient scripting (See also ticket 2782)
icecast	N	Y	Deploy certs to Icecast 2 streaming media servers
pritunl	N	Y	Install certs in pritunl distributed OpenVPN servers
proxmox	N	Y	Install certs in Proxmox Virtualization servers
postfix	N	Y	STARTTLS Everywhere is becoming a Certbot Postfix/Exim plugin

If you're interested, you can also *write your own plugin*.

Re-running Certbot

Running Certbot with the `certonly` or `run` commands always requests the creation of a single new certificate, even if you already have an existing certificate with some of the same domain names. The `--force-renewal`, `--duplicate`, and `--expand` options control Certbot's behavior in this case. If you don't specify a requested behavior, Certbot may ask you what you intended.

`--force-renewal` tells Certbot to request a new certificate with the same domains as an existing certificate. (Each and every domain must be explicitly specified via `-d`.) If successful, this certificate will be saved alongside the earlier one and symbolic links (the "`live`" reference) will be updated to point to the new certificate. This is a valid method of explicitly requesting the renewal of a specific individual certificate.

`--duplicate` tells Certbot to create a separate, unrelated certificate with the same domains as an existing certificate. This certificate will be saved completely separately from the prior one. Most users probably do not want this behavior.

`--expand` tells Certbot to update an existing certificate with a new certificate that contains all of the old domains and one or more additional new domains.

`--allow-subset-of-names` tells Certbot to continue with cert generation if only some of the specified domain authorazations can be obtained. This may be useful if some domains specified in a certificate no longer point at this system.

Whenever you obtain a new certificate in any of these ways, the new certificate exists alongside any previously-obtained certificates, whether or not the previous certificates have expired. The generation of a new certificate counts against several rate limits that are intended to prevent abuse of the ACME protocol, as described here.

Certbot also provides a `renew` command. This command examines *all* existing certificates to determine whether or not each is near expiry. For any existing certificate that is near expiry, `certbot renew` will attempt to obtain a new certificate for the same domains. Unlike `certonly`, `renew` acts on multiple certificates and always takes into account whether each one is near expiry. Because of this, `renew` is suitable (and designed) for automated use, to allow your system to automatically renew each certificate when appropriate. Since `renew` will only renew certificates that are near expiry it can be run as frequently as you want - since it will usually take no action.

Typically, `certbot renew` runs a reduced risk of rate-limit problems because it renews certificates only when necessary, and because some of the Let's Encrypt CA's rate limit policies treat the issuance of a new certificate under these circumstances more generously. More details about the use of `certbot renew` are provided below.

Renewing certificates

Note: Let's Encrypt CA issues short-lived certificates (90 days). Make sure you renew the certificates at least once in 3 months.

The `certbot` client now supports a `renew` action to check all installed certificates for impending expiry and attempt to renew them. The simplest form is simply

```
certbot renew
```

This will attempt to renew any previously-obtained certificates that expire in less than 30 days. The same plugin and options that were used at the time the certificate was originally issued will be used for the renewal attempt, unless you specify other plugins or options.

You can also specify hooks to be run before or after a certificate is renewed. For example, if you have only a single cert and you obtained it using the *standalone* plugin, it will be used by default when renewing. In that case you may want to use a command like this to renew your certificate.

```
certbot renew --pre-hook "service nginx stop" --post-hook "service nginx
start"
```

This will stop Nginx so standalone can bind to the necessary ports and then restart Nginx after the plugin is finished. The hooks will only be run if a certificate is due for renewal, so you can run this command frequently without unnecessarily stopping your webserver. More information about renewal hooks can be found by running `certbot --help renew`.

If you're sure that this command executes successfully without human intervention, you can add the command to `crontab` (since certificates are only renewed when they're determined to be near expiry, the command can run on a regular basis, like every week or every day). In that case, you are likely to want to use the `-q` or `--quiet` quiet flag to silence all output except errors.

If you are manually renewing all of your certificates, the `--force-renewal` flag may be helpful; it causes the expiration time of the certificate(s) to be ignored when considering renewal, and attempts to renew each and every installed certificate regardless of its age. (This form is not appropriate to run daily because each certificate will be renewed every day, which will quickly run into the certificate authority rate limit.)

Note that options provided to `certbot renew` will apply to *every* certificate for which renewal is attempted; for example, `certbot renew --rsa-key-size 4096` would try to replace every near-expiry certificate with an equivalent certificate using a 4096-bit RSA public key. If a certificate is successfully renewed using specified options, those options will be saved and used for future renewals of that certificate.

An alternative form that provides for more fine-grained control over the renewal process (while renewing specified certificates one at a time), is `certbot certonly` with the complete set of subject domains of a specific certificate specified via `-d` flags. You may also want to include the `-n` or `--noninteractive` flag to prevent blocking on user input (which is useful when running the command from cron).

`certbot certonly -n -d example.com -d www.example.com`

(All of the domains covered by the certificate must be specified in this case in order to renew and replace the old certificate rather than obtaining a new one; don't forget any `www.` domains! Specifying a subset of the domains creates a new, separate certificate containing only those domains, rather than replacing the original certificate.) When run with a set of domains corresponding to an existing certificate, the `certonly` command attempts to renew that one individual certificate.

Please note that the CA will send notification emails to the address you provide if you do not renew certificates that are about to expire.

Certbot is working hard on improving the renewal process, and we apologize for any inconveniences you encounter in integrating these commands into your individual environment.

Certbot command-line options

Certbot supports a lot of command line options. Here's the full list, from `certbot --help all`:

```
usage:
  certbot [SUBCOMMAND] [options] [-d domain] [-d domain] ...

Certbot can obtain and install HTTPS/TLS/SSL certificates.  By default,
it will attempt to use a webserver both for obtaining and installing the
cert. Major SUBCOMMANDS are:

  (default) run        Obtain & install a cert in your current webserver
  certonly             Obtain cert, but do not install it (aka "auth")
  install              Install a previously obtained cert in a server
  renew                Renew previously obtained certs that are near expiry
  revoke               Revoke a previously obtained certificate
  register             Perform tasks related to registering with the CA
  rollback             Rollback server configuration changes made during install
  config_changes       Show changes made to server config during installation
  plugins              Display information about installed plugins

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILE, --config CONFIG_FILE
                        config file path (default: None)
  -v, --verbose         This flag can be used multiple times to incrementally
                        increase the verbosity of output, e.g. -vvv. (default:
                        -2)
  -t, --text            Use the text output instead of the curses UI.
                        (default: False)
  -n, --non-interactive, --noninteractive
                        Run without ever asking for user input. This may
                        require additional command line flags; the client will
                        try to explain which ones are required if it finds one
                        missing (default: False)
  --dialog              Run using interactive dialog menus (default: False)
  -d DOMAIN, --domains DOMAIN, --domain DOMAIN
                        Domain names to apply. For multiple domains you can
                        use multiple -d flags or enter a comma separated list
                        of domains as a parameter. (default: [])
  --dry-run             Perform a test run of the client, obtaining test
                        (invalid) certs but not saving them to disk. This can
                        currently only be used with the 'certonly' and 'renew'
                        subcommands. Note: Although --dry-run tries to avoid
                        making any persistent changes on a system, it is not
                        completely side-effect free: if used with webserver
                        authenticator plugins like apache and nginx, it makes
                        and then reverts temporary config changes in order to
                        obtain test certs, and reloads webservers to deploy
                        and then roll back those changes. It also calls --pre-
                        hook and --post-hook commands if they are defined
                        because they may be necessary to accurately simulate
                        renewal. --renew-hook commands are not called.
                        (default: False)
  --register-unsafely-without-email
                        Specifying this flag enables registering an account
                        with no email address. This is strongly discouraged,
                        because in the event of key loss or account compromise
                        you will irrevocably lose access to your account. You
                        will also be unable to receive notice about impending
                        expiration or revocation of your certificates. Updates
                        to the Subscriber Agreement will still affect you, and
                        will be effective 14 days after posting an update to
                        the web site. (default: False)
  --update-registration
                        With the register verb, indicates that details
                        associated with an existing registration, such as the
                        e-mail address, should be updated, rather than
                        registering a new account. (default: False)
  -m EMAIL, --email EMAIL
                        Email used for registration and recovery contact.
                        (default: None)
  --preferred-challenges PREF_CHALLS
                        A sorted, comma delimited list of the preferred
                        challenge to use during authorization with the most
                        preferred challenge listed first (Eg, "dns" or "tls-
                        sni-01,http,dns"). Not all plugins support all
                        challenges. See
                        https://certbot.eff.org/docs/using.html#plugins for
                        details. ACME Challenges are versioned, but if you
                        pick "http" rather than "http-01", Certbot will select
                        the latest version automatically. (default: [])
  --user-agent USER_AGENT
                        Set a custom user agent string for the client. User
                        agent strings allow the CA to collect high level
                        statistics about success rates by OS and plugin. If
                        you wish to hide your server OS version from the Let's
                        Encrypt server, set this to "". (default: None)

automation:
  Arguments for automating execution & other tweaks

  --keep-until-expiring, --keep, --reinstall
                        If the requested cert matches an existing cert, always
                        keep the existing one until it is due for renewal (for
                        the 'run' subcommand this means reinstall the existing
                        cert) (default: False)
  --expand              If an existing cert covers some subset of the
                        requested names, always expand and replace it with the
                        additional names. (default: False)
  --version             show program's version number and exit
  --force-renewal, --renew-by-default
                        If a certificate already exists for the requested
                        domains, renew it now, regardless of whether it is
                        near expiry. (Often --keep-until-expiring is more
                        appropriate). Also implies --expand. (default: False)
  --allow-subset-of-names
                        When performing domain validation, do not consider it
                        a failure if authorizations can not be obtained for a
                        strict subset of the requested domains. This may be
                        useful for allowing renewals for multiple domains to
                        succeed even if some domains no longer point at this
                        system. This option cannot be used with --csr.
                        (default: False)
  --agree-tos           Agree to the ACME Subscriber Agreement (default:
                        False)
  --account ACCOUNT_ID  Account ID to use (default: None)
  --duplicate           Allow making a certificate lineage that duplicates an
                        existing one (both can be renewed in parallel)
                        (default: False)
  --os-packages-only    (certbot-auto only) install OS package dependencies
                        and then stop (default: False)
  --no-self-upgrade     (certbot-auto only) prevent the certbot-auto script
                        from upgrading itself to newer released versions
                        (default: False)
  -q, --quiet           Silence all output except errors. Useful for
                        automation via cron. Implies --non-interactive.
                        (default: False)

security:
  Security parameters & server settings

  --rsa-key-size N      Size of the RSA key. (default: 2048)
  --must-staple         Adds the OCSP Must Staple extension to the
                        certificate. Autoconfigures OCSP Stapling for
                        supported setups (Apache version >= 2.3.3 ). (default:
                        False)
  --redirect            Automatically redirect all HTTP traffic to HTTPS for
                        the newly authenticated vhost. (default: None)
  --no-redirect         Do not automatically redirect all HTTP traffic to
                        HTTPS for the newly authenticated vhost. (default:
                        None)
  --hsts                Add the Strict-Transport-Security header to every HTTP
                        response. Forcing browser to always use SSL for the
                        domain. Defends against SSL Stripping. (default:
                        False)
  --no-hsts             Do not automatically add the Strict-Transport-Security
                        header to every HTTP response. (default: False)
  --uir                 Add the "Content-Security-Policy: upgrade-insecure-
                        requests" header to every HTTP response. Forcing the
                        browser to use https:// for every http:// resource.
                        (default: None)
  --no-uir              Do not automatically set the "Content-Security-Policy:
                        upgrade-insecure-requests" header to every HTTP
                        response. (default: None)
  --staple-ocsp         Enables OCSP Stapling. A valid OCSP response is
                        stapled to the certificate that the server offers
                        during TLS. (default: None)
  --no-staple-ocsp      Do not automatically enable OCSP Stapling. (default:
                        None)
  --strict-permissions  Require that all configuration files are owned by the
                        current user; only needed if your config is somewhere
                        unsafe like /tmp/ (default: False)

testing:
  The following flags are meant for testing purposes only! Do NOT change
  them, unless you really know what you're doing!

  --test-cert, --staging
                        Use the staging server to obtain test (invalid) certs;
                        equivalent to --server https://acme-
                        staging.api.letsencrypt.org/directory (default: False)
  --debug               Show tracebacks in case of errors, and allow certbot-
                        auto execution on experimental platforms (default:
                        False)
  --no-verify-ssl       Disable verification of the ACME server's certificate.
                        (default: False)
  --break-my-certs      Be willing to replace or renew valid certs with
                        invalid (testing/staging) certs (default: False)

renew:
  The 'renew' subcommand will attempt to renew all certificates (or more
  precisely, certificate lineages) you have previously obtained if they are
  close to expiry, and print a summary of the results. By default, 'renew'
  will reuse the options used to create obtain or most recently successfully
  renew each certificate lineage. You can try it with `--dry-run` first. For
  more fine-grained control, you can renew individual lineages with the
  `certonly` subcommand. Hooks are available to run commands before and
  after renewal; see https://certbot.eff.org/docs/using.html#renewal for
  more information on these.

  --pre-hook PRE_HOOK   Command to be run in a shell before obtaining any
                        certificates. Intended primarily for renewal, where it
                        can be used to temporarily shut down a webserver that
                        might conflict with the standalone plugin. This will
                        only be called if a certificate is actually to be
                        obtained/renewed. (default: None)
  --post-hook POST_HOOK
                        Command to be run in a shell after attempting to
                        obtain/renew certificates. Can be used to deploy
                        renewed certificates, or to restart any servers that
                        were stopped by --pre-hook. This is only run if an
                        attempt was made to obtain/renew a certificate.
                        (default: None)
  --renew-hook RENEW_HOOK
                        Command to be run in a shell once for each
                        successfully renewed certificate. For this command,
                        the shell variable $RENEWED_LINEAGE will point to the
                        config live subdirectory containing the new certs and
                        keys; the shell variable $RENEWED_DOMAINS will contain
                        a space-delimited list of renewed cert domains
                        (default: None)
  --disable-hook-validation
                        Ordinarily the commands specified for --pre-hook
                        /--post-hook/--renew-hook will be checked for
                        validity, to see if the programs being run are in the
                        $PATH, so that mistakes can be caught early, even when
                        the hooks aren't being run just yet. The validation is
                        rather simplistic and fails if you use more advanced
                        shell constructs, so you can use this switch to
                        disable it. (default: True)

certonly:
  Options for modifying how a cert is obtained

  --tls-sni-01-port TLS_SNI_01_PORT
                        Port used during tls-sni-01 challenge. This only
                        affects the port Certbot listens on. A conforming ACME
                        server will still attempt to connect on port 443.
                        (default: 443)
  --http-01-port HTTP01_PORT
                        Port used in the http-01 challenge.This only affects
                        the port Certbot listens on. A conforming ACME server
                        will still attempt to connect on port 80. (default:
                        80)
  --csr CSR             Path to a Certificate Signing Request (CSR) in DER or
                        PEM format. Currently --csr only works with the
                        'certonly' subcommand. (default: None)

install:
  Options for modifying how a cert is deployed

revoke:
  Options for revocation of certs

rollback:
  Options for reverting config changes

  --checkpoints N       Revert configuration N number of checkpoints.
                        (default: 1)

plugins:
  Options for the "plugins" subcommand

  --init                Initialize plugins. (default: False)
  --prepare             Initialize and prepare plugins. (default: False)
  --authenticators      Limit to authenticator plugins only. (default: None)
  --installers          Limit to installer plugins only. (default: None)

config_changes:
  Options for showing a history of config changes

  --num NUM             How many past revisions you want to be displayed
                        (default: None)

paths:
  Arguments changing execution paths & servers

  --cert-path CERT_PATH
                        Path to where cert is saved (with auth --csr),
                        installed from or revoked. (default: None)
  --key-path KEY_PATH   Path to private key for cert installation or
                        revocation (if account key is missing) (default: None)
  --fullchain-path FULLCHAIN_PATH
                        Accompanying path to a full certificate chain (cert
                        plus chain). (default: None)
  --chain-path CHAIN_PATH
                        Accompanying path to a certificate chain. (default:
                        None)
  --config-dir CONFIG_DIR
                        Configuration directory. (default: /etc/letsencrypt)
  --work-dir WORK_DIR   Working directory. (default: /var/lib/letsencrypt)
  --logs-dir LOGS_DIR   Logs directory. (default: /var/log/letsencrypt)
  --server SERVER       ACME Directory Resource URI. (default:
                        https://acme-v01.api.letsencrypt.org/directory)

plugins:
  Plugin Selection: Certbot client supports an extensible plugins
  architecture. See 'certbot plugins' for a list of all installed plugins
  and their names. You can force a particular plugin by setting options
  provided below. Running --help <plugin_name> will list flags specific to
  that plugin.

  -a AUTHENTICATOR, --authenticator AUTHENTICATOR
                        Authenticator plugin name. (default: None)
  -i INSTALLER, --installer INSTALLER
                        Installer plugin name (also used to find domains).
                        (default: None)
  --configurator CONFIGURATOR
                        Name of the plugin that is both an authenticator and
                        an installer. Should not be used together with
                        --authenticator or --installer. (default: None)
  --apache              Obtain and install certs using Apache (default: False)
  --nginx               Obtain and install certs using Nginx (default: False)
  --standalone          Obtain certs using a "standalone" webserver. (default:
                        False)
  --manual              Provide laborious manual instructions for obtaining a
                        cert (default: False)
  --webroot             Obtain certs by placing files in a webroot directory.
                        (default: False)

nginx:
  Nginx Web Server plugin - Alpha
  --nginx-server-root NGINX_SERVER_ROOT
                        Nginx server root directory. (default: /etc/nginx)
  --nginx-ctl NGINX_CTL
                        Path to the 'nginx' binary, used for 'configtest' and
                        retrieving nginx version number. (default: nginx)

standalone:
  Spin up a temporary webserver

manual:
  Manually configure an HTTP server

  --manual-test-mode    Test mode. Executes the manual command in subprocess.
                        (default: False)
  --manual-public-ip-logging-ok
                        Automatically allows public IP logging. (default:
                        False)

webroot:
  Place files in webroot directory

  --webroot-path WEBROOT_PATH, -w WEBROOT_PATH
                        public_html / webroot path. This can be specified
                        multiple times to handle different domains; each
                        domain will have the webroot path that preceded it.
                        For instance: `-w /var/www/example -d example.com -d
                        www.example.com -w /var/www/thing -d thing.net -d
                        m.thing.net` (default: [])
  --webroot-map WEBROOT_MAP
                        JSON dictionary mapping domains to webroot paths; this
                        implies -d for each entry. You may need to escape this
                        from your shell. E.g.: --webroot-map
                        '{"eg1.is,m.eg1.is":"/www/eg1/", "eg2.is":"/www/eg2"}'
                        This option is merged with, but takes precedence over,
                        -w / -d entries. At present, if you put webroot-map in
                        a config file, it needs to be on a single line, like:
                        webroot-map = {"example.com":"/var/www"}. (default:
                        {})

apache:
  Apache Web Server plugin - Beta

  --apache-enmod APACHE_ENMOD
                        Path to the Apache 'a2enmod' binary. (default:
                        a2enmod)
  --apache-dismod APACHE_DISMOD
                        Path to the Apache 'a2dismod' binary. (default:
                        a2dismod)
  --apache-le-vhost-ext APACHE_LE_VHOST_EXT
                        SSL vhost configuration extension. (default: -le-
                        ssl.conf)
  --apache-server-root APACHE_SERVER_ROOT
                        Apache server root directory. (default: /etc/apache2)
  --apache-vhost-root APACHE_VHOST_ROOT
                        Apache server VirtualHost configuration root (default:
                        /etc/apache2/sites-available)
  --apache-logs-root APACHE_LOGS_ROOT
```

```
                        Apache server logs directory (default:
                        /var/log/apache2)
  --apache-challenge-location APACHE_CHALLENGE_LOCATION
                        Directory path for challenge configuration. (default:
                        /etc/apache2)
  --apache-handle-modules APACHE_HANDLE_MODULES
                        Let installer handle enabling required modules for
                        you.(Only Ubuntu/Debian currently) (default: True)
  --apache-handle-sites APACHE_HANDLE_SITES
                        Let installer handle enabling sites for you.(Only
                        Ubuntu/Debian currently) (default: True)

null:
  Null Installer
```

Where are my certificates?

All generated keys and issued certificates can be found in `/etc/letsencrypt/live/$domain`. Rather than copying, please point your (web) server configuration directly to those files (or create symlinks). During the *renewal*, `/etc/letsencrypt/live` is updated with the latest necessary files.

Note: `/etc/letsencrypt/archive` and `/etc/letsencrypt/keys` contain all previous keys and certificates, while `/etc/letsencrypt/live` symlinks to the latest versions.

The following files are available:

`privkey.pem` Private key for the certificate.

> **Warning:** This **must be kept secret at all times**! Never share it with anyone, including Certbot developers. You cannot put it into a safe, however - your server still needs to access this file in order for SSL/TLS to work.

This is what Apache needs for SSLCertificateKeyFile, and Nginx for ssl_certificate_key.

`fullchain.pem` All certificates, **including** server certificate (aka leaf certificate or end-entity certificate). The server certificate is the first one in this file, followed by any intermediates.

This is what Apache >= 2.4.8 needs for SSLCertificateFile, and what Nginx needs for ssl_certificate.

`cert.pem` and `chain.pem` (less common) `cert.pem` contains the server certificate by itself, and `chain.pem` contains the additional intermediate certificate or certificates that web browsers will need in order to validate the server certificate. If you provide one of these files to your web server, you **must** provide both of them, or some browsers will show "This Connection is Untrusted" errors for your site, some of the time.

Apache < 2.4.8 needs these for SSLCertificateFile. and SSLCertificateChainFile, respectively.

If you're using OCSP stapling with Nginx >= 1.3.7, `chain.pem` should be provided as the ssl_trusted_certificate to validate OCSP responses.

Note: All files are PEM-encoded. If you need other format, such as DER or PFX, then you could convert using `openssl`. You can automate that with `--renew-hook` if you're using automatic *renewal*.

Configuration file

It is possible to specify configuration file with `certbot-auto --config cli.ini` (or shorter `-c cli.ini`). An example configuration file is shown below:

```
# This is an example of the kind of things you can do in a configuration file.
# All flags used by the client can be configured here. Run Certbot with
# "--help" to learn more about the available options.

# Use a 4096 bit RSA key instead of 2048
rsa-key-size = 4096

# Uncomment and update to register with the specified e-mail address
# email = foo@example.com

# Uncomment and update to generate certificates for the specified
# domains.
# domains = example.com, www.example.com

# Uncomment to use a text interface instead of ncurses
# text = True

# Uncomment to use the standalone authenticator on port 443
# authenticator = standalone
# standalone-supported-challenges = tls-sni-01

# Uncomment to use the webroot authenticator. Replace webroot-path with the
# path to the public_html / webroot folder being served by your web server.
# authenticator = webroot
# webroot-path = /usr/share/nginx/html
```

By default, the following locations are searched:

- `/etc/letsencrypt/cli.ini`
- `$XDG_CONFIG_HOME/letsencrypt/cli.ini` (or `~/.config/letsencrypt/cli.ini` if `$XDG_CONFIG_HOME` is not set).

Getting help

If you're having problems, we recommend posting on the Let's Encrypt Community Forum.

You can also chat with us on IRC: (#certbot @ OFTC) or (#letsencrypt @ freenode).

If you find a bug in the software, please do report it in our issue tracker. Remember to give us as much information as possible:

- copy and paste exact command line used and the output (though mind that the latter might include some personally identifiable information, including your email and domains)
- copy and paste logs from `/var/log/letsencrypt` (though mind they also might contain personally identifiable information)
- copy and paste `certbot --version` output
- your operating system, including specific version
- specify which installation method you've chosen

CHAPTER

FOUR

DEVELOPER GUIDE

Table of Contents

Getting Started

Running a local copy of the client

Running the client in developer mode from your local tree is a little different than running `letsencrypt-auto`. To get set up, do these things once:

```
git clone https://github.com/certbot/certbot
cd certbot
./letsencrypt-auto-source/letsencrypt-auto --os-packages-only
./tools/venv.sh
```

Then in each shell where you're working on the client, do:

```
source ./venv/bin/activate
```

After that, your shell will be using the virtual environment, and you run the client by typing:

```
certbot
```

Activating a shell in this way makes it easier to run unit tests with `tox` and integration tests, as described below. To reverse this, you can type `deactivate`. More information can be found in the virtualenv docs.

Find issues to work on

You can find the open issues in the github issue tracker. Comparatively easy ones are marked Good Volunteer Task. If you're starting work on something, post a comment to let others know and seek feedback on your plan where appropriate.

Once you've got a working branch, you can open a pull request. All changes in your pull request must have thorough unit test coverage, pass our *integration* tests, and be compliant with the *coding style*.

Testing

The following tools are there to help you:

- `tox` starts a full set of tests. Please note that it includes apacheconftest, which uses the system's Apache install to test config file parsing, so it should only be run on systems that have an experimental, non-production Apache2 install on them. `tox -e apacheconftest` can be used to run those specific Apache conf tests.
- `tox --skip-missing-interpreters` runs tox while ignoring missing versions of Python needed for running the tests.
- `tox -e py27`, `tox -e py26` etc, run unit tests for specific Python versions.
- `tox -e cover` checks the test coverage only. Calling the `./tox.cover.sh` script directly (or even `./tox.cover.sh $pkg1 $pkg2 ...` for any subpackages) might be a bit quicker, though.
- `tox -e lint` checks the style of the whole project, while `pylint --rcfile=.pylintrc path` will check a single file or specific directory only.
- For debugging, we recommend `pip install ipdb` and putting `import ipdb; ipdb.set_trace()` statement inside the source code. Alternatively, you can use Python's standard library pdb, but you won't get TAB completion...

Integration testing with the boulder CA

Generally it is sufficient to open a pull request and let Github and Travis run integration tests for you.

However, if you prefer to run tests, you can use Vagrant, using the Vagrantfile in Certbot's repository. To execute the tests on a Vagrant box, the only command you are required to run is:

```
./tests/boulder-integration.sh
```

Otherwise, please follow the following instructions.

Mac OS X users: Run `./tests/mac-bootstrap.sh` instead of `boulder-start.sh` to install dependencies, configure the environment, and start boulder.

Otherwise, install Go 1.5, `libtool-ltdl`, `mariadb-server` and `rabbitmq-server` and then start Boulder, an ACME CA server.

If you can't get packages of Go 1.5 for your Linux system, you can execute the following commands to install it:

```
wget https://storage.googleapis.com/golang/go1.5.3.linux-amd64.tar.gz -P /tmp/
sudo tar -C /usr/local -xzf /tmp/go1.5.3.linux-amd64.tar.gz
if ! grep -Fxq "export GOROOT=/usr/local/go" ~/.profile ; then echo "export GOROOT=/
↪usr/local/go" >> ~/.profile; fi
if ! grep -Fxq "export PATH=\\$GOROOT/bin:\\$PATH" ~/.profile ; then echo "export␣
↪PATH=\\$GOROOT/bin:\\$PATH" >> ~/.profile; fi
```

These commands download Go 1.5.3 to `/tmp/`, extracts to `/usr/local`, and then adds the export lines required to execute `boulder-start.sh` to `~/.profile` if they were not previously added

Make sure you execute the following command after Go finishes installing:

```
if ! grep -Fxq "export GOPATH=\\$HOME/go" ~/.profile ; then echo "export GOPATH=\\
↪$HOME/go" >> ~/.profile; fi
```

Afterwards, you'd be able to start Boulder using the following command:

```
./tests/boulder-start.sh
```

The script will download, compile and run the executable; please be patient - it will take some time... Once its ready, you will see `Server running, listening on 127.0.0.1:4000...`. Add `/etc/hosts` entries pointing `le.wtf`, `le1.wtf`, `le2.wtf`, `le3.wtf` and `nginx.wtf` to 127.0.0.1. You may now run (in a separate terminal):

```
./tests/boulder-integration.sh && echo OK || echo FAIL
```

If you would like to test `certbot_nginx` plugin (highly encouraged) make sure to install prerequisites as listed in `certbot-nginx/tests/boulder-integration.sh` and rerun the integration tests suite.

Code components and layout

acme contains all protocol specific code

certbot all client code

Plugin-architecture

Certbot has a plugin architecture to facilitate support for different webservers, other TLS servers, and operating systems. The interfaces available for plugins to implement are defined in interfaces.py and plugins/common.py.

The most common kind of plugin is a "Configurator", which is likely to implement the *`IAuthenticator`* and *`IInstaller`* interfaces (though some Configurators may implement just one of those).

There are also *`IDisplay`* plugins, which implement bindings to alternative UI libraries.

Authenticators

Authenticators are plugins designed to prove that this client deserves a certificate for some domain name by solving challenges received from the ACME server. From the protocol, there are essentially two different types of challenges. Challenges that must be solved by individual plugins in order to satisfy domain validation (subclasses of `DVChallenge`, i.e. `TLSSNI01`, `HTTP01`, `DNS`) and continuity specific challenges (subclasses of `ContinuityChallenge`, i.e. `RecoveryToken`, `RecoveryContact`, `ProofOfPossession`). Continuity challenges are always handled by the `ContinuityAuthenticator`, while plugins are expected to handle `DVChallenge` types. Right now, we have two authenticator plugins, the `ApacheConfigurator` and the `StandaloneAuthenticator`. The Standalone and Apache authenticators only solve the `TLSSNI01` challenge currently. (You can set which challenges your authenticator can handle through the *`get_chall_pref()`*.

(FYI: We also have a partial implementation for a `DNSAuthenticator` in a separate branch).

Installer

Installers plugins exist to actually setup the certificate in a server, possibly tweak the security configuration to make it more correct and secure (Fix some mixed content problems, turn on HSTS, redirect to HTTPS, etc). Installer plugins tell the main client about their abilities to do the latter via the *`supported_enhancements()`* call. We currently have two Installers in the tree, the `ApacheConfigurator`. and the `NginxConfigurator`. External projects have made some progress toward support for IIS, Icecast and Plesk.

Installers and Authenticators will oftentimes be the same class/object (because for instance both tasks can be performed by a webserver like nginx) though this is not always the case (the standalone plugin is an authenticator that listens on port 443, but it cannot install certs; a postfix plugin would be an installer but not an authenticator).

Installers and Authenticators are kept separate because it should be possible to use the `StandaloneAuthenticator` (it sets up its own Python server to perform challenges) with a program that cannot solve challenges itself (Such as MTA installers).

Installer Development

There are a few existing classes that may be beneficial while developing a new *`IInstaller`*. Installers aimed to reconfigure UNIX servers may use Augeas for configuration parsing and can inherit from `AugeasConfigurator` class to handle much of the interface. Installers that are unable to use Augeas may still find the *`Reverter`* class helpful in handling configuration checkpoints and rollback.

Display

We currently offer a pythondialog and "text" mode for displays. Display plugins implement the *`IDisplay`* interface.

Writing your own plugin

Certbot client supports dynamic discovery of plugins through the setuptools entry points. This way you can, for example, create a custom implementation of `IAuthenticator` or the `IInstaller` without having to merge it with the core upstream source code. An example is provided in `examples/plugins/` directory.

Warning: Please be aware though that as this client is still in a developer-preview stage, the API may undergo a few changes. If you believe the plugin will be beneficial to the community, please consider submitting a pull request to the repo and we will update it with any necessary API changes.

Coding style

Please:

1. **Be consistent with the rest of the code**.
2. Read PEP 8 - Style Guide for Python Code.
3. Follow the Google Python Style Guide, with the exception that we use Sphinx-style documentation:

```
def foo(arg):
    """Short description.

    :param int arg: Some number.

    :returns: Argument
    :rtype: int

    """
    return arg
```

4. Remember to use `pylint`.

Submitting a pull request

Steps:

1. Write your code!
2. Make sure your environment is set up properly and that you're in your virtualenv. You can do this by running `./tools/venv.sh`. (this is a **very important** step)
3. Run `./pep8.travis.sh` to do a cursory check of your code style. Fix any errors.
4. Run `tox -e lint` to check for pylint errors. Fix any errors.
5. Run `tox --skip-missing-interpreters` to run the entire test suite including coverage. The `--skip-missing-interpreters` argument ignores missing versions of Python needed for running the tests. Fix any errors.
6. If your code touches communication with an ACME server/Boulder, you should run the integration tests, see *integration*. See **'Known Issues'_** for some common failures that have nothing to do with your code.
7. Submit the PR.

8. Did your tests pass on Travis? If they didn't, fix any errors.

Updating the documentation

In order to generate the Sphinx documentation, run the following commands:

```
make -C docs clean html man
```

This should generate documentation in the `docs/_build/html` directory.

Other methods for running the client

Vagrant

If you are a Vagrant user, Certbot comes with a Vagrantfile that automates setting up a development environment in an Ubuntu 14.04 LTS VM. To set it up, simply run `vagrant up`. The repository is synced to `/vagrant`, so you can get started with:

```
vagrant ssh
cd /vagrant
sudo ./venv/bin/certbot
```

Support for other Linux distributions coming soon.

Note: Unfortunately, Python distutils and, by extension, setup.py and tox, use hard linking quite extensively. Hard linking is not supported by the default sync filesystem in Vagrant. As a result, all actions with these commands are *significantly slower* in Vagrant. One potential fix is to use NFS (related issue).

Docker

OSX users will probably find it easiest to set up a Docker container for development. Certbot comes with a Dockerfile (`Dockerfile-dev`) for doing so. To use Docker on OSX, install and setup docker-machine using the instructions at https://docs.docker.com/installation/mac/.

To build the development Docker image:

```
docker build -t certbot -f Dockerfile-dev .
```

Now run tests inside the Docker image:

```
docker run -it certbot bash
cd src
tox -e py27
```

Notes on OS dependencies

OS-level dependencies can be installed like so:

```
letsencrypt-auto-source/letsencrypt-auto --os-packages-only
```

In general...

- `sudo` is required as a suggested way of running privileged process
- Python 2.6/2.7 is required
- Augeas is required for the Python bindings
- `virtualenv` and `pip` are used for managing other python library dependencies

Debian

For squeeze you will need to:

- Use `virtualenv --no-site-packages -p python` instead of `-p python2`.

FreeBSD

Package installation for FreeBSD uses `pkg`, not ports.

FreeBSD by default uses `tcsh`. In order to activate virtualenv (see below), you will need a compatible shell, e.g. `pkg install bash && bash`.

CHAPTER

FIVE

PACKAGING GUIDE

Releases

We release packages and upload them to PyPI (wheels and source tarballs).

- https://pypi.python.org/pypi/acme
- https://pypi.python.org/pypi/certbot
- https://pypi.python.org/pypi/certbot-apache
- https://pypi.python.org/pypi/certbot-nginx

The following scripts are used in the process:

- https://github.com/letsencrypt/letsencrypt/blob/master/tools/release.sh

We currently version with the following scheme:

- `0.1.0`
- `0.2.0dev` for developement in `master`
- `0.2.0` (only temporarily in `master`)
- ...

Notes for package maintainers

0. Please use our releases, not `master`!
1. Do not package `certbot-compatibility-test` or `letshelp-certbot` - it's only used internally.
2. If you'd like to include automated renewal in your package `certbot renew -q` should be added to crontab or systemd timer.
3. `jws` is an internal script for `acme` module and it doesn't have to be packaged - it's mostly for debugging: you can use it as `echo foo | jws sign | jws verify`.
4. Do get in touch with us. We are happy to make any changes that will make packaging easier. If you need to apply some patches don't do it downstream - make a PR here.

Already ongoing efforts

Arch

From our official releases: - https://www.archlinux.org/packages/community/any/python2-acme - https://www.archlinux.org/packages/community/any/certbot - https://www.archlinux.org/packages/community/any/certbot-apache - https://www.archlinux.org/packages/community/any/certbot-nginx - https://www.archlinux.org/packages/community/any/letshelp-certbot

From `master`: https://aur.archlinux.org/packages/certbot-git

Debian (and its derivatives, including Ubuntu)

https://packages.debian.org/sid/certbot https://packages.debian.org/sid/python-certbot https://packages.debian.org/sid/python-certbot-apache

Fedora

In Fedora 23+.

- https://admin.fedoraproject.org/pkgdb/package/letsencrypt/
- https://admin.fedoraproject.org/pkgdb/package/certbot/
- https://admin.fedoraproject.org/pkgdb/package/python-acme/

FreeBSD

https://svnweb.freebsd.org/ports/head/security/py-certbot/

GNU Guix

- https://www.gnu.org/software/guix/package-list.html#certbot

OpenBSD

- http://cvsweb.openbsd.org/cgi-bin/cvsweb/ports/security/letsencrypt/client/

RESOURCES

Documentation: https://certbot.eff.org/docs

Software project: https://github.com/certbot/certbot

Notes for developers: https://certbot.eff.org/docs/contributing.html

Main Website: https://certbot.eff.org

Let's Encrypt Website: https://letsencrypt.org

IRC Channel: #letsencrypt on Freenode or #certbot on OFTC

Community: https://community.letsencrypt.org

ACME spec: http://ietf-wg-acme.github.io/acme/

ACME working area in github: https://github.com/ietf-wg-acme/acme

Mailing list: client-dev (to subscribe without a Google account, send an email to client-dev+subscribe@letsencrypt.org)

CHAPTER

SEVEN

API DOCUMENTATION

`certbot.account`

Creates ACME accounts for server.

class `certbot.account.`**`Account`**(*regr*, *key*, *meta=None*)

Bases: `object`

ACME protocol registration.

Variables

- **`regr`** (*`RegistrationResource`*) – Registration Resource
- **`key`** (*`JWK`*) – Authorized Account Key
- ***`Meta`*** – Account metadata
- **`id`** (*`str`*) – Globally unique account identifier.

class **`Meta`**(***kwargs*)

Bases: `acme.jose.json_util.JSONObjectWithFields`

Account metadata

Variables

- **`creation_dt`** (*`datetime.datetime`*) – Creation date and time (UTC).
- **`creation_host`** (*`str`*) – FQDN of host, where account has been created.

Note: `creation_dt` and `creation_host` are useful in cross-machine migration scenarios.

`Account.`**`slug`**

Short account identification string, useful for UI.

`certbot.account.`**`report_new_account`**(*acc*, *config*)

Informs the user about their new ACME account.

class `certbot.account.`**`AccountMemoryStorage`**(*initial_accounts=None*)

Bases: *`certbot.interfaces.AccountStorage`*

In-memory account strage.

class `certbot.account.`**`AccountFileStorage`**(*config*)

Bases: *`certbot.interfaces.AccountStorage`*

Accounts file storage.

Variables **config** (IConfig) – Client configuration

save_regr (*account*)
Save the registration resource.

Parameters **account** (Account) – account whose regr should be saved

certbot.achallenges

Client annotated ACME challenges.

Please use names such as `achall` to distiguish from variables "of type" `acme.challenges.Challenge` (denoted by `chall`) and `ChallengeBody` (denoted by `challb`):

```
from acme import challenges
from acme import messages
from certbot import achallenges

chall = challenges.DNS(token='foo')
challb = messages.ChallengeBody(chall=chall)
achall = achallenges.DNS(chall=challb, domain='example.com')
```

Note, that all annotated challenges act as a proxy objects:

```
achall.token == challb.token
```

class certbot.achallenges.**AnnotatedChallenge** (***kwargs*)
Bases: `acme.jose.util.ImmutableMap`

Client annotated challenge.

Wraps around server provided challenge and annotates with data useful for the client.

Variables **challb** – Wrapped `ChallengeBody`.

class certbot.achallenges.**KeyAuthorizationAnnotatedChallenge** (***kwargs*)
Bases: *certbot.achallenges.AnnotatedChallenge*

Client annotated `KeyAuthorizationChallenge` challenge.

response_and_validation (**args*, ***kwargs*)
Generate response and validation.

class certbot.achallenges.**DNS** (***kwargs*)
Bases: *certbot.achallenges.AnnotatedChallenge*

Client annotated "dns" ACME challenge.

acme_type
alias of *DNS*

certbot.auth_handler

ACME AuthHandler.

class certbot.auth_handler.**AuthHandler** (*auth*, *acme*, *account*, *pref_challs*)
Bases: `object`

ACME Authorization Handler for a client.

Variables

- **auth** – Authenticator capable of solving `Challenge` types
- **acme** (*acme.client.Client*) – ACME client API.
- ***account*** – Client's Account
- **authzr** (*dict*) – ACME Authorization Resource dict where keys are domains and values are `acme.messages.AuthorizationResource`
- **achalls** (*list*) – DV challenges in the form of *certbot.achallenges.AnnotatedChallenge*
- **pref_challs** (*list*) – sorted user specified preferred challenges in the form of subclasses of `acme.challenges.Challenge` with the most preferred challenge listed first

get_authorizations(*domains*, *best_effort=False*)
Retrieve all authorizations for challenges.

Parameters

- **domains** (*list*) – Domains for authorization
- **best_effort** (*bool*) – Whether or not all authorizations are required (this is useful in renewal)

Returns List of authorization resources

Return type list

Raises ***AuthorizationError*** – If unable to retrieve all authorizations

_choose_challenges(*domains*)
Retrieve necessary challenges to satisfy server.

_solve_challenges()
Get Responses for challenges from authenticators.

_respond(*resp*, *best_effort*)
Send/Receive confirmation of all challenges.

Note: This method also cleans up the auth_handler state.

_send_responses(*achalls*, *resps*, *chall_update*)
Send responses and make sure errors are handled.

Parameters **chall_update** (*dict*) – parameter that is updated to hold authzr -> list of outstanding solved annotated challenges

_poll_challenges(*chall_update*, *best_effort*, *min_sleep=3*, *max_rounds=15*)
Wait for all challenge results to be determined.

_handle_check(*domain*, *achalls*)
Returns tuple of ('completed', 'failed').

_find_updated_challb(*authzr*, *achall*)
Find updated challenge body within Authorization Resource.

Warning: This assumes only one instance of type of challenge in each challenge resource.

Parameters

- **authzr** (*AuthorizationResource*) – Authorization Resource
- **achall** (AnnotatedChallenge) – Annotated challenge for which to get status

_get_chall_pref(*domain*)
Return list of challenge preferences.

Parameters **domain** (*str*) – domain for which you are requesting preferences

_cleanup_challenges(*achall_list=None*)
Cleanup challenges.

If achall_list is not provided, cleanup all achallenges.

verify_authzr_complete()
Verifies that all authorizations have been decided.

Returns Whether all authzr are complete

Return type bool

_challenge_factory(*domain*, *path*)
Construct Namedtuple Challenges

Parameters

- **domain** (*str*) – domain of the enrollee
- **path** (*list*) – List of indices from `challenges`.

Returns achalls, list of challenge type `certbot.achallenges.Indexed`

Return type list

Raises ***errors.Error*** – if challenge type is not recognized

certbot.auth_handler.**challb_to_achall**(*challb*, *account_key*, *domain*)
Converts a ChallengeBody object to an AnnotatedChallenge.

Parameters

- **challb** (*ChallengeBody*) – ChallengeBody
- **account_key** (*JWK*) – Authorized Account Key
- **domain** (*str*) – Domain of the challb

Returns Appropriate AnnotatedChallenge

Return type *certbot.achallenges.AnnotatedChallenge*

certbot.auth_handler.**gen_challenge_path**(*challbs*, *preferences*, *combinations*)
Generate a plan to get authority over the identity.

Todo

This can be possibly be rewritten to use resolved_combinations.

Parameters

- **challbs** (*tuple*) – A tuple of challenges (`acme.messages.Challenge`) from `acme.messages.AuthorizationResource` to be fulfilled by the client in order to prove possession of the identifier.

- **preferences** (*list*) – List of challenge preferences for domain (`acme.challenges.Challenge` subclasses)
- **combinations** (*tuple*) – A collection of sets of challenges from `acme.messages.Challenge`, each of which would be sufficient to prove possession of the identifier.

Returns tuple of indices from `challenges`.

Return type tuple

Raises ***certbot.errors.AuthorizationError*** – If a path cannot be created that satisfies the CA given the preferences and combinations.

`certbot.auth_handler.`**`_find_smart_path`**(*challbs*, *preferences*, *combinations*)
Find challenge path with server hints.

Can be called if combinations is included. Function uses a simple ranking system to choose the combo with the lowest cost.

`certbot.auth_handler.`**`_find_dumb_path`**(*challbs*, *preferences*)
Find challenge path without server hints.

Should be called if the combinations hint is not included by the server. This function either returns a path containing all challenges provided by the CA or raises an exception.

`certbot.auth_handler.`**`_report_no_chall_path`**()
Logs and raises an error that no satisfiable chall path exists.

`certbot.auth_handler.`**`_report_failed_challs`**(*failed_achalls*)
Notifies the user about failed challenges.

Parameters **failed_achalls** (*set*) – A set of failed *certbot.achallenges.AnnotatedChallenge*.

`certbot.auth_handler.`**`_generate_failed_chall_msg`**(*failed_achalls*)
Creates a user friendly error message about failed challenges.

Parameters **failed_achalls** (*list*) – A list of failed *certbot.achallenges.AnnotatedChallenge* with the same error type.

Returns A formatted error message for the client.

Return type str

certbot.client

Certbot client API.

`certbot.client.`**`acme_from_config_key`**(*config*, *key*)
Wrangle ACME client construction

`certbot.client.`**`_determine_user_agent`**(*config*)
Set a user_agent string in the config based on the choice of plugins. (this wasn't knowable at construction time)

Returns the client's User-Agent string

Return type `str`

`certbot.client.`**`register`**(*config*, *account_storage*, *tos_cb=None*)
Register new account with an ACME CA.

This function takes care of generating fresh private key, registering the account, optionally accepting CA Terms of Service and finally saving the account. It should be called prior to initialization of *`Client`*, unless account has already been created.

Parameters

- **`config`** (`IConfig`) – Client configuration.
- **`account_storage`** (`AccountStorage`) – Account storage where newly registered account will be saved to. Save happens only after TOS acceptance step, so any account private keys or `RegistrationResource` will not be persisted if `tos_cb` returns `False`.
- **`tos_cb`** – If ACME CA requires the user to accept a Terms of Service before registering account, client action is necessary. For example, a CLI tool would prompt the user acceptance. `tos_cb` must be a callable that should accept `RegistrationResource` and return a `bool`: `True` iff the Terms of Service present in the contained `Registration.terms_of_service` is accepted by the client, and `False` otherwise. `tos_cb` will be called only if the client acction is necessary, i.e. when `terms_of_service is not None`. This argument is optional, if not supplied it will default to automatic acceptance!

Raises

- ***`certbot.errors.Error`*** – In case of any client problems, in particular registration failure, or unaccepted Terms of Service.
- **`acme.errors.Error`** – In case of any protocol problems.

Returns Newly registered and saved account, as well as protocol API handle (should be used in *`Client`* initialization).

Return type `tuple` of *`Account`* and `acme.client.Client`

`certbot.client.`**`perform_registration`**(*acme*, *config*)

Actually register new account, trying repeatedly if there are email problems

Parameters

- **`config`** (`IConfig`) – Client configuration.
- **`client`** (*`acme.client.Client`*) – ACME client object.

Returns Registration Resource.

Return type `acme.messages.RegistrationResource`

Raises **`UnexpectedUpdate`** –

class `certbot.client.`**`Client`**(*config*, *account_*, *auth*, *installer*, *acme=None*)

Bases: `object`

ACME protocol client.

Variables

- **`config`** (`IConfig`) – Client configuration.
- ***`account`*** (`Account`) – Account registered with *`register`*.
- ***`auth_handler`*** (`AuthHandler`) – Authorizations handler that will dispatch DV challenges to appropriate authenticators (providing *`IAuthenticator`* interface).
- **`auth`** (`IAuthenticator`) – Prepared (`IAuthenticator.prepare`) authenticator that can solve ACME challenges.
- **`installer`** (`IInstaller`) – Installer.

- **acme** (*acme.client.Client*) – Optional ACME client API handle. You might already have one from *register*.

obtain_certificate_from_csr (*domains*, *csr*, *typ=2*, *authzr=None*)
Obtain certificate.

Internal function with precondition that `domains` are consistent with identifiers present in the `csr`.

Parameters

- **domains** (*list*) – Domain names.
- **csr** (`util.CSR`) – DER-encoded Certificate Signing Request. The key used to generate this CSR can be different than `authkey`.
- **authzr** (*list*) – List of `acme.messages.AuthorizationResource`

Returns `CertificateResource` and certificate chain (as returned by `fetch_chain`).

Return type tuple

obtain_certificate (*domains*)
Obtains a certificate from the ACME server.

register must be called before *obtain_certificate*

Parameters **domains** (*list*) – domains to get a certificate

Returns `CertificateResource`, certificate chain (as returned by `fetch_chain`), and newly generated private key (*util.Key*) and DER-encoded Certificate Signing Request (*util.CSR*).

Return type tuple

obtain_and_enroll_certificate (*domains*)
Obtain and enroll certificate.

Get a new certificate for the specified domains using the specified authenticator and installer, and then create a new renewable lineage containing it.

Parameters

- **domains** (*list*) – Domains to request.
- **plugins** – A PluginsFactory object.

Returns A new *certbot.storage.RenewableCert* instance referred to the enrolled cert lineage, False if the cert could not be obtained, or None if doing a successful dry run.

save_certificate (*certr*, *chain_cert*, *cert_path*, *chain_path*, *fullchain_path*)
Saves the certificate received from the ACME server.

Parameters

- **certr** (`acme.messages.Certificate`) – ACME "certificate" resource.
- **chain_cert** (*list*) –
- **cert_path** (*str*) – Candidate path to a certificate.
- **chain_path** (*str*) – Candidate path to a certificate chain.
- **fullchain_path** (*str*) – Candidate path to a full cert chain.

Returns cert_path, chain_path, and fullchain_path as absolute paths to the actual files

Return type `tuple` of `str`

Raises `IOError` – If unable to find room to write the cert files

`deploy_certificate`(*domains*, *privkey_path*, *cert_path*, *chain_path*, *fullchain_path*)
Install certificate

Parameters

- **`domains`** (`list`) – list of domains to install the certificate
- **`privkey_path`** (`str`) – path to certificate private key
- **`cert_path`** (`str`) – certificate file path (optional)
- **`chain_path`** (`str`) – chain file path

`enhance_config`(*domains*, *config*, *chain_path*)
Enhance the configuration.

Parameters

- **`domains`** (`list`) – list of domains to configure
- **`chain_path`** (`str` or `None`) – chain file path

Variables **`config`** – Namespace typically produced by `argparse.ArgumentParser.parse_args()`. it must have the redirect, hsts and uir attributes.

Raises ***`errors.Error`*** – if no installer is specified in the client.

`apply_enhancement`(*domains*, *enhancement*, *options=None*)
Applies an enhacement on all domains.

Parameters **`domains`** – list of ssl_vhosts

:type list of str

Parameters **`enhancement`** – name of enhancement, e.g. ensure-http-header

:type str

Note: when more options are need make options a list.

Parameters **`options`** – options to enhancement, e.g. Strict-Transport-Security

:type str

Raises ***`errors.PluginError`*** – If Enhancement is not supported, or if there is any other problem with the enhancement.

`_recovery_routine_with_msg`(*success_msg*)
Calls the installer's recovery routine and prints success_msg

Parameters **`success_msg`** (`str`) – message to show on successful recovery

`_rollback_and_restart`(*success_msg*)
Rollback the most recent checkpoint and restart the webserver

Parameters **`success_msg`** (`str`) – message to show on successful rollback

`certbot.client.`**`validate_key_csr`**(*privkey*, *csr=None*)
Validate Key and CSR files.

Verifies that the client key and csr arguments are valid and correspond to one another. This does not currently check the names in the CSR due to the inability to read SANs from CSRs in python crypto libraries.

If csr is left as None, only the key will be validated.

Parameters

- **privkey** (*certbot.util.Key*) – Key associated with CSR
- **csr** (util.CSR) – CSR

Raises ***errors.Error*** – when validation fails

certbot.client.**rollback**(*default_installer*, *checkpoints*, *config*, *plugins*)

Revert configuration the specified number of checkpoints.

Parameters

- **checkpoints** (*int*) – Number of checkpoints to revert.
- **config** (*certbot.interfaces.IConfig*) – Configuration.

certbot.client.**view_config_changes**(*config*, *num=None*)

View checkpoints and associated configuration changes.

Note: This assumes that the installation is using a Reverter object.

Parameters **config** (*certbot.interfaces.IConfig*) – Configuration.

certbot.client.**_open_pem_file**(*cli_arg_path*, *pem_path*)

Open a pem file.

If cli_arg_path was set by the client, open that. Otherwise, uniquify the file path.

Parameters

- **cli_arg_path** (*str*) – the cli arg name, e.g. cert_path
- **pem_path** (*str*) – the pem file path to open

Returns a tuple of file object and its absolute file path

certbot.client.**_save_chain**(*chain_pem*, *chain_file*)

Saves chain_pem at a unique path based on chain_path.

Parameters

- **chain_pem** (*str*) – certificate chain in PEM format
- **chain_file** (*str*) – chain file object

certbot.configuration

Certbot user-supplied configuration.

class certbot.configuration.**NamespaceConfig**(*namespace*)

Bases: object

Configuration wrapper around argparse.Namespace.

For more documentation, including available attributes, please see *certbot.interfaces.IConfig*. However, note that the following attributes are dynamically resolved using *work_dir* and relative paths defined in *certbot.constants*:

- accounts_dir

- `csr_dir`
- `in_progress_dir`
- `key_dir`
- `renewer_config_file`
- `temp_checkpoint_dir`

Variables **`namespace`** – Namespace typically produced by `argparse.ArgumentParser.parse_args()`.

`server_path`
File path based on `server`.

class `certbot.configuration.`**`RenewerConfiguration`**(*namespace*)
Bases: `object`

Configuration wrapper for renewer.

`certbot.configuration.`**`check_config_sanity`**(*config*)
Validate command line options and display error message if requirements are not met.

Parameters **`config`** – IConfig instance holding user configuration

certbot.constants

Certbot constants.

`certbot.constants.`**`SETUPTOOLS_PLUGINS_ENTRY_POINT`** **= 'certbot.plugins'**
Setuptools entry point group name for plugins.

`certbot.constants.`**`OLD_SETUPTOOLS_PLUGINS_ENTRY_POINT`** **= 'letsencrypt.plugins'**
Plugins Setuptools entry point before rename.

`certbot.constants.`**`STAGING_URI`** **= 'https://acme-staging.api.letsencrypt.org/directory'**
Defaults for CLI flags and *`IConfig`* attributes.

`certbot.constants.`**`QUIET_LOGGING_LEVEL`** **= 30**
Logging level to use in quiet mode.

`certbot.constants.`**`RENEWER_DEFAULTS`** **= {'renew_before_expiry': '30 days', 'deploy_before_expiry': '99 years', 'renew**
Defaults for renewer script.

`certbot.constants.`**`ENHANCEMENTS`** **= ['redirect', 'http-header', 'ocsp-stapling', 'spdy']**
List of possible *`certbot.interfaces.IInstaller`* enhancements.

List of expected options parameters: - redirect: None - http-header: TODO - ocsp-stapling: certificate chain file path - spdy: TODO

`certbot.constants.`**`ARCHIVE_DIR`** **= 'archive'**
Archive directory, relative to `IConfig.config_dir`.

`certbot.constants.`**`CONFIG_DIRS_MODE`** **= 493**
Directory mode for `.IConfig.config_dir` et al.

`certbot.constants.`**`ACCOUNTS_DIR`** **= 'accounts'**
Directory where all accounts are saved.

`certbot.constants.`**`BACKUP_DIR`** **= 'backups'**
Directory (relative to `IConfig.work_dir`) where backups are kept.

certbot.constants.**CSR_DIR** = **'csr'**
See *IConfig.csr_dir*.

certbot.constants.**IN_PROGRESS_DIR** = **'IN_PROGRESS'**
Directory used before a permanent checkpoint is finalized (relative to `IConfig.work_dir`).

certbot.constants.**KEY_DIR** = **'keys'**
Directory (relative to `IConfig.config_dir`) where keys are saved.

certbot.constants.**LIVE_DIR** = **'live'**
Live directory, relative to `IConfig.config_dir`.

certbot.constants.**TEMP_CHECKPOINT_DIR** = **'temp_checkpoint'**
Temporary checkpoint directory (relative to `IConfig.work_dir`).

certbot.constants.**RENEWAL_CONFIGS_DIR** = **'renewal'**
Renewal configs directory, relative to `IConfig.config_dir`.

certbot.constants.**RENEWER_CONFIG_FILENAME** = **'renewer.conf'**
Renewer config file name (relative to `IConfig.config_dir`).

certbot.crypto_util

Certbot client crypto utility functions.

Todo

Make the transition to use PSS rather than PKCS1_v1_5 when the server is capable of handling the signatures.

certbot.crypto_util.**init_save_key**(*key_size*, *key_dir*, *keyname='key-certbot.pem'*)
Initializes and saves a privkey.

Inits key and saves it in PEM format on the filesystem.

Note: keyname is the attempted filename, it may be different if a file already exists at the path.

Parameters

- **key_size** (*int*) – RSA key size in bits
- **key_dir** (*str*) – Key save directory.
- **keyname** (*str*) – Filename of key

Returns Key

Return type *certbot.util.Key*

Raises **ValueError** – If unable to generate the key given key_size.

certbot.crypto_util.**init_save_csr**(*privkey*, *names*, *path*, *csrname='csr-certbot.pem'*)
Initialize a CSR with the given private key.

Parameters

- **privkey** (*certbot.util.Key*) – Key to include in the CSR
- **names** (*set*) – `str` names to include in the CSR

- **path** (*str*) – Certificate save directory.

Returns CSR

Return type *certbot.util.CSR*

`certbot.crypto_util.`**`make_csr`**(*key_str*, *domains*, *must_staple=False*)

Generate a CSR.

Parameters

- **key_str** (*str*) – PEM-encoded RSA key.
- **domains** (*list*) – Domains included in the certificate.

Todo

Detect duplicates in `domains`? Using a set doesn't preserve order...

Returns new CSR in PEM and DER form containing all domains

Return type tuple

`certbot.crypto_util.`**`valid_csr`**(*csr*)

Validate CSR.

Check if `csr` is a valid CSR for the given domains.

Parameters **csr** (*str*) – CSR in PEM.

Returns Validity of CSR.

Return type bool

`certbot.crypto_util.`**`csr_matches_pubkey`**(*csr*, *privkey*)

Does private key correspond to the subject public key in the CSR?

Parameters

- **csr** (*str*) – CSR in PEM.
- **privkey** (*str*) – Private key file contents (PEM)

Returns Correspondence of private key to CSR subject public key.

Return type bool

`certbot.crypto_util.`**`import_csr_file`**(*csrfile*, *data*)

Import a CSR file, which can be either PEM or DER.

Parameters

- **csrfile** (*str*) – CSR filename
- **data** (*str*) – contents of the CSR file

Returns (`OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`, util.CSR object representing the CSR, list of domains requested in the CSR)

Return type tuple

`certbot.crypto_util.`**`make_key`**(*bits*)

Generate PEM encoded RSA key.

Parameters **bits** (*int*) – Number of bits, at least 1024.

Returns new RSA key in PEM form with specified number of bits

Return type str

`certbot.crypto_util.`**`valid_privkey`**(*privkey*)

Is valid RSA private key?

Parameters **`privkey`** (*`str`*) – Private key file contents in PEM

Returns Validity of private key.

Return type bool

`certbot.crypto_util.`**`pyopenssl_load_certificate`**(*data*)

Load PEM/DER certificate.

Raises ***`errors.Error`*** –

`certbot.crypto_util.`**`get_sans_from_cert`**(*cert*, *typ=1*)

Get a list of Subject Alternative Names from a certificate.

Parameters

- **`cert`** (*`str`*) – Certificate (encoded).
- **`typ`** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of Subject Alternative Names.

Return type list

`certbot.crypto_util.`**`get_sans_from_csr`**(*csr*, *typ=1*)

Get a list of Subject Alternative Names from a CSR.

Parameters

- **`csr`** (*`str`*) – CSR (encoded).
- **`typ`** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of Subject Alternative Names.

Return type list

`certbot.crypto_util.`**`get_names_from_cert`**(*csr*, *typ=1*)

Get a list of domains from a cert, including the CN if it is set.

Parameters

- **`cert`** (*`str`*) – Certificate (encoded).
- **`typ`** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of domain names.

Return type list

`certbot.crypto_util.`**`get_names_from_csr`**(*csr*, *typ=1*)

Get a list of domains from a CSR, including the CN if it is set.

Parameters

- **`csr`** (*`str`*) – CSR (encoded).

- **typ** – `OpenSSL.crypto.FILETYPE_PEM` or `OpenSSL.crypto.FILETYPE_ASN1`

Returns A list of domain names.

Return type list

`certbot.crypto_util.`**`dump_pyopenssl_chain`**(*chain*, *filetype=1*)

Dump certificate chain into a bundle.

Parameters **chain** (*list*) – List of `OpenSSL.crypto.X509` (or wrapped in `acme.jose.ComparableX509`).

`certbot.crypto_util.`**`notBefore`**(*cert_path*)

When does the cert at cert_path start being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the notBefore value from the cert at cert_path

Return type `datetime.datetime`

`certbot.crypto_util.`**`notAfter`**(*cert_path*)

When does the cert at cert_path stop being valid?

Parameters **cert_path** (*str*) – path to a cert in PEM format

Returns the notAfter value from the cert at cert_path

Return type `datetime.datetime`

`certbot.crypto_util.`**`_notAfterBefore`**(*cert_path*, *method*)

Internal helper function for finding notbefore/notafter.

Parameters

- **cert_path** (*str*) – path to a cert in PEM format
- **method** (*function*) – one of `OpenSSL.crypto.X509.get_notBefore` or `OpenSSL.crypto.X509.get_notAfter`

Returns the notBefore or notAfter value from the cert at cert_path

Return type `datetime.datetime`

certbot.display

Certbot display utilities.

certbot.display.util

Certbot display.

`certbot.display.util.`**`OK`** **= 'ok'**

Display exit code indicating user acceptance.

`certbot.display.util.`**`CANCEL`** **= 'cancel'**

Display exit code for a user canceling the display.

`certbot.display.util.`**`HELP`** **= 'help'**

Display exit code when for when the user requests more help.

`certbot.display.util.`**`ESC`** **= 'esc'**

Display exit code when the user hits Escape

`certbot.display.util.`**`_wrap_lines`**(*msg*)

Format lines nicely to 80 chars.

Parameters **`msg`** (*`str`*) – Original message

Returns Formatted message respecting newlines in message

Return type str

class `certbot.display.util.`**`FileDisplay`**(*outfile*)

Bases: `object`

File-based display.

`notification`(*message*, *pause=True*)

Displays a notification and waits for user acceptance.

Parameters

- **`message`** (*`str`*) – Message to display
- **`pause`** (*`bool`*) – Whether or not the program should pause for the user's confirmation

`menu`(*message*, *choices*, *ok_label=''*, *cancel_label=''*, *help_label=''*, ***unused_kwargs*)

Display a menu.

Todo

This doesn't enable the help label/button (I wasn't sold on any interface I came up with for this). It would be a nice feature

Parameters

- **`message`** (*`str`*) – title of menu
- **`choices`** (*`list of tuples (tag, item) or list of descriptions (tags will be enumerated)`*) – Menu lines, len must be > 0
- **`_kwargs`** (*`dict`*) – absorbs default / cli_args

Returns tuple of (`code`, `index`) where `code` - str display exit code `index` - int index of the user's selection

Return type tuple

`input`(*message*, ***unused_kwargs*)

Accept input from the user.

Parameters

- **`message`** (*`str`*) – message to display to the user
- **`_kwargs`** (*`dict`*) – absorbs default / cli_args

Returns tuple of (`code`, *`input`*) where `code` - str display exit code *`input`* - str of the user's input

Return type tuple

yesno (*message*, *yes_label='Yes'*, *no_label='No'*, ***unused_kwargs*)
Query the user with a yes/no question.

Yes and No label must begin with different letters, and must contain at least one letter each.

Parameters

- **message** (*str*) – question for the user
- **yes_label** (*str*) – Label of the "Yes" parameter
- **no_label** (*str*) – Label of the "No" parameter
- **_kwargs** (*dict*) – absorbs default / cli_args

Returns True for "Yes", False for "No"

Return type bool

checklist (*message*, *tags*, *default_status=True*, ***unused_kwargs*)
Display a checklist.

Parameters

- **message** (*str*) – Message to display to user
- **tags** (*list*) – `str` tags to select, len(tags) > 0
- **default_status** (*bool*) – Not used for FileDisplay
- **_kwargs** (*dict*) – absorbs default / cli_args

Returns tuple of (`code`, `tags`) where `code` - str display exit code `tags` - list of selected tags

Return type tuple

directory_select (*message*, ***unused_kwargs*)
Display a directory selection screen.

Parameters **message** (*str*) – prompt to give the user

Returns tuple of the form (`code`, `string`) where `code` - display exit code `string` - input entered by the user

_scrub_checklist_input (*indices*, *tags*)
Validate input and transform indices to appropriate tags.

Parameters

- **indices** (*list*) – input
- **tags** (*list*) – Original tags of the checklist

Returns valid tags the user selected

Return type `list` of `str`

_print_menu (*message*, *choices*)
Print a menu on the screen.

Parameters

- **message** (*str*) – title of menu
- **choices** (*list of tuples (tag, item) or list of descriptions (tags will be enumerated)*) – Menu lines

_get_valid_int_ans (*max_*)
Get a numerical selection.

Parameters **`max`** (*`int`*) – The maximum entry (len of choices), must be positive

Returns tuple of the form (`code`, `selection`) where `code` - str display exit code ('ok' or cancel') `selection` - int user's selection

Return type tuple

class `certbot.display.util.`**`NoninteractiveDisplay`**(*outfile*)

Bases: `object`

An iDisplay implementation that never asks for interactive user input

`_interaction_fail`(*message*, *cli_flag*, *extra=''*)

Error out in case of an attempt to interact in noninteractive mode

`notification`(*message*, *pause=False*)

Displays a notification without waiting for user acceptance.

Parameters

- **`message`** (*`str`*) – Message to display to stdout
- **`pause`** (*`bool`*) – The NoninteractiveDisplay waits for no keyboard

`menu`(*message*, *choices*, *ok_label=None*, *cancel_label=None*, *help_label=None*, *default=None*, *cli_flag=None*)

Avoid displaying a menu.

Parameters

- **`choices`** (*`list of tuples (tag, item) or list of descriptions (tags will be enumerated)`*) – Menu lines, len must be > 0
- **`message`** (*`str`*) – title of menu
- **`default`** (*`int`*) – the default choice
- **`kwargs`** (*`dict`*) – absorbs various irrelevant labelling arguments

Returns tuple of (`code`, `index`) where `code` - str display exit code `index` - int index of the user's selection

Return type tuple

Raises ***`errors.MissingCommandlineFlag`*** – if there was no default

`input`(*message*, *default=None*, *cli_flag=None*)

Accept input from the user.

Parameters **`message`** (*`str`*) – message to display to the user

Returns tuple of (`code`, *`input`*) where `code` - str display exit code *`input`* - str of the user's input

Return type tuple

Raises ***`errors.MissingCommandlineFlag`*** – if there was no default

`yesno`(*message*, *yes_label=None*, *no_label=None*, *default=None*, *cli_flag=None*)

Decide Yes or No, without asking anybody

Parameters

- **`message`** (*`str`*) – question for the user
- **`kwargs`** (*`dict`*) – absorbs yes_label, no_label

Raises ***`errors.MissingCommandlineFlag`*** – if there was no default

Returns True for “Yes”, False for “No”

Return type bool

checklist (*message*, *tags*, *default=None*, *cli_flag=None*, ***kwargs*)
Display a checklist.

Parameters

- **message** (*str*) – Message to display to user
- **tags** (*list*) – str tags to select, len(tags) > 0
- **kwargs** (*dict*) – absorbs default_status arg

Returns tuple of (code, tags) where code - str display exit code tags - list of selected tags

Return type tuple

directory_select (*message*, *default=None*, *cli_flag=None*)
Simulate prompting the user for a directory.

This function returns default if it is not None, otherwise, an exception is raised explaining the problem. If cli_flag is not None, the error message will include the flag that can be used to set this value with the CLI.

Parameters

- **message** (*str*) – prompt to give the user
- **default** – default value to return (if one exists)
- **cli_flag** (*str*) – option used to set this value with the CLI

Returns tuple of the form (code, string) where code - int display exit code string - input entered by the user

certbot.display.util.**separate_list_input** (*input_*)
Separate a comma or space separated list.

Parameters input (*str*) – input from the user

Returns strings

Return type list

certbot.display.util.**_parens_around_char** (*label*)
Place parens around first character of label.

Parameters label (*str*) – Must contain at least one character

certbot.display.ops

Contains UI methods for LE user operations.

certbot.display.ops.**get_email** (*invalid=False*, *optional=True*)
Prompt for valid email address.

Parameters

- **invalid** (*bool*) – True if an invalid address was provided by the user
- **optional** (*bool*) – True if the user can use –register-unsafely-without-email to avoid providing an e-mail

Returns e-mail address

Return type str

Raises *errors.Error* – if the user cancels

`certbot.display.ops.`**`choose_account`**(*accounts*)
Choose an account.

Parameters **`accounts`** (*list*) – Containing at least one *Account*

`certbot.display.ops.`**`choose_names`**(*installer*)
Display screen to select domains to validate.

Parameters **`installer`** (*certbot.interfaces.IInstaller*) – An installer object

Returns List of selected names

Return type `list` of `str`

`certbot.display.ops.`**`get_valid_domains`**(*domains*)

Helper method for choose_names that implements basic checks on domain names

Parameters **`domains`** (*list*) – Domain names to validate

Returns List of valid domains

Return type list

`certbot.display.ops.`**`_filter_names`**(*names*)
Determine which names the user would like to select from a list.

Parameters **`names`** (*list*) – domain names

Returns tuple of the form (`code`, `names`) where `code` - str display exit code `names` - list of names selected

Return type tuple

`certbot.display.ops.`**`_choose_names_manually`**(*prompt_prefix=''*)
Manually input names for those without an installer.

Parameters **`prompt_prefix`** (*str*) – string to prepend to prompt for domains

Returns list of provided names

Return type `list` of `str`

`certbot.display.ops.`**`success_installation`**(*domains*)
Display a box confirming the installation of HTTPS.

Todo

This should be centered on the screen

Parameters **`domains`** (*list*) – domain names which were enabled

`certbot.display.ops.`**`success_renewal`**(*domains*, *action*)
Display a box confirming the renewal of an existing certificate.

Todo

This should be centered on the screen

Parameters

- **domains** (*list*) – domain names which were renewed
- **action** (*str*) – can be "reinstall" or "renew"

certbot.display.ops.**_gen_ssl_lab_urls**(*domains*)

Returns a list of urls.

Parameters **domains** (*list*) – Each domain is a 'str'

certbot.display.ops.**_gen_https_names**(*domains*)

Returns a string of the https domains.

Domains are formatted nicely with https:// prepended to each.

Parameters **domains** (*list*) – Each domain is a 'str'

certbot.display.enhancements

Certbot Enhancement Display

certbot.display.enhancements.**ask**(*enhancement*)

Display the enhancement to the user.

Parameters **enhancement** (*str*) – One of the certbot.CONFIG.ENHANCEMENTS enhancements

Returns True if feature is desired, False otherwise

Return type bool

Raises ***errors.Error*** – if the enhancement provided is not supported

certbot.display.enhancements.**redirect_by_default**()

Determines whether the user would like to redirect to HTTPS.

Returns True if redirect is desired, False otherwise

Return type bool

certbot.errors

Certbot client errors.

exception certbot.errors.**Error**

Bases: exceptions.Exception

Generic Certbot client error.

exception certbot.errors.**AccountStorageError**

Bases: *certbot.errors.Error*

Generic *AccountStorage* error.

exception certbot.errors.**AccountNotFound**

Bases: *certbot.errors.AccountStorageError*

Account not found error.

exception `certbot.errors.`**`ReverterError`**
Bases: *`certbot.errors.Error`*

Certbot Reverter error.

exception `certbot.errors.`**`SubprocessError`**
Bases: *`certbot.errors.Error`*

Subprocess handling error.

exception `certbot.errors.`**`CertStorageError`**
Bases: *`certbot.errors.Error`*

Generic `CertStorage` error.

exception `certbot.errors.`**`HookCommandNotFound`**
Bases: *`certbot.errors.Error`*

Failed to find a hook command in the PATH.

exception `certbot.errors.`**`SignalExit`**
Bases: *`certbot.errors.Error`*

A Unix signal was recieved while in the ErrorHandler context manager.

exception `certbot.errors.`**`AuthorizationError`**
Bases: *`certbot.errors.Error`*

Authorization error.

exception `certbot.errors.`**`FailedChallenges`** (*failed_achalls*)
Bases: *`certbot.errors.AuthorizationError`*

Failed challenges error.

Variables **`failed_achalls`** (*`set`*) – Failed *`AnnotatedChallenge`* instances.

exception `certbot.errors.`**`PluginError`**
Bases: *`certbot.errors.Error`*

Certbot Plugin error.

exception `certbot.errors.`**`PluginEnhancementAlreadyPresent`**
Bases: *`certbot.errors.Error`*

Enhancement was already set

exception `certbot.errors.`**`PluginSelectionError`**
Bases: *`certbot.errors.Error`*

A problem with plugin/configurator selection or setup

exception `certbot.errors.`**`NoInstallationError`**
Bases: *`certbot.errors.PluginError`*

Certbot No Installation error.

exception `certbot.errors.`**`MisconfigurationError`**
Bases: *`certbot.errors.PluginError`*

Certbot Misconfiguration error.

exception `certbot.errors.`**`NotSupportedError`**
Bases: *`certbot.errors.PluginError`*

Certbot Plugin function not supported error.

exception `certbot.errors.`**`StandaloneBindError`**(*socket_error*, *port*)
Bases: *`certbot.errors.Error`*

Standalone plugin bind error.

exception `certbot.errors.`**`ConfigurationError`**
Bases: *`certbot.errors.Error`*

Configuration sanity error.

exception `certbot.errors.`**`MissingCommandlineFlag`**
Bases: *`certbot.errors.Error`*

A command line argument was missing in noninteractive usage

certbot

Certbot client.

certbot.interfaces

Certbot client interfaces.

class `certbot.interfaces.`**`AccountStorage`**
Bases: `object`

Accounts storage interface.

`find_all`()
Find all accounts.

Returns All found accounts.

Return type list

`load`(*account_id*)
Load an account by its id.

Raises

- ***`AccountNotFound`*** – if account could not be found
- ***`AccountStorageError`*** – if account could not be loaded

`save`(*account*)
Save account.

Raises ***`AccountStorageError`*** – if account could not be saved

interface `certbot.interfaces.`**`IPluginFactory`**
IPlugin factory.

Objects providing this interface will be called without satisfying any entry point "extras" (extra dependencies) you might have defined for your plugin, e.g (excerpt from `setup.py` script):

```
setup(
    ...
    entry_points={
        'certbot.plugins': [
            'name=example_project.plugin[plugin_deps]',
        ],
    },
    extras_require={
        'plugin_deps': ['dep1', 'dep2'],
    }
)
```

Therefore, make sure such objects are importable and usable without extras. This is necessary, because CLI does the following operations (in order):

- loads an entry point,
- calls *inject_parser_options*,
- requires an entry point,
- creates plugin instance (*__call__*).

description
Short plugin description

__call__(*config*, *name*)
Create new *IPlugin*.

Parameters

- **config** (IConfig) – Configuration.
- **name** (*str*) – Unique plugin name.

inject_parser_options(*parser*, *name*)
Inject argument parser options (flags).

1. Be nice and prepend all options and destinations with *option_namespace* and `dest_namespace`.
2. Inject options (flags) only. Positional arguments are not allowed, as this would break the CLI.

Parameters

- **parser** (*ArgumentParser*) – (Almost) top-level CLI parser.
- **name** (*str*) – Unique plugin name.

interface `certbot.interfaces.`**IPlugin**
Certbot plugin.

prepare()
Prepare the plugin.

Finish up any additional initialization.

Raises

- ***PluginError*** – when full initialization cannot be completed.
- ***MisconfigurationError*** – when full initialization cannot be completed. Plugin will be displayed on a list of available plugins.
- ***NoInstallationError*** – when the necessary programs/files cannot be located. Plugin will NOT be displayed on a list of available plugins.
- ***NotSupportedError*** – when the installation is recognized, but the version is not currently supported.

more_info()

Human-readable string to help the user.

Should describe the steps taken and any relevant info to help the user decide which plugin to use.

Rtype str

interface `certbot.interfaces.`**`IAuthenticator`**

Extends: *`certbot.interfaces.IPlugin`*

Generic Certbot Authenticator.

Class represents all possible tools processes that have the ability to perform challenges and attain a certificate.

get_chall_pref(*domain*)

Return list of challenge preferences.

Parameters **domain** (*str*) – Domain for which challenge preferences are sought.

Returns List of challenge types (subclasses of `acme.challenges.Challenge`) with the most preferred challenges first. If a type is not specified, it means the Authenticator cannot perform the challenge.

Return type list

perform(*achalls*)

Perform the given challenge.

Parameters **achalls** (*list*) – Non-empty (guaranteed) list of *`AnnotatedChallenge`* instances, such that it contains types found within *`get_chall_pref()`* only.

Returns

List of ACME `ChallengeResponse` instances or if the `Challenge` cannot be fulfilled then:

`None` Authenticator can perform challenge, but not at this time.

`False` Authenticator will never be able to perform (error).

Return type `list` of `acme.challenges.ChallengeResponse`, where responses are required to be returned in the same order as corresponding input challenges

Raises ***`PluginError`*** – If challenges cannot be performed

cleanup(*achalls*)

Revert changes and shutdown after challenges complete.

This method should be able to revert all changes made by perform, even if perform exited abnormally.

Parameters **achalls** (*list*) – Non-empty (guaranteed) list of *`AnnotatedChallenge`* instances, a subset of those previously passed to *`perform()`*.

Raises ***`PluginError`*** – if original configuration cannot be restored

interface `certbot.interfaces.`**`IConfig`**

Certbot user-supplied configuration.

Warning: The values stored in the configuration have not been filtered, stripped or sanitized.

server

ACME Directory Resource URI.

`email`
: Email used for registration and recovery contact.

`rsa_key_size`
: Size of the RSA key.

`must_staple`
: Adds the OCSP Must Staple extension to the certificate. Autoconfigures OCSP Stapling for supported setups (Apache version >= 2.3.3).

`config_dir`
: Configuration directory.

`work_dir`
: Working directory.

`accounts_dir`
: Directory where all account information is stored.

`backup_dir`
: Configuration backups directory.

`csr_dir`
: Directory where newly generated Certificate Signing Requests (CSRs) are saved.

`in_progress_dir`
: Directory used before a permanent checkpoint is finalized.

`key_dir`
: Keys storage.

`temp_checkpoint_dir`
: Temporary checkpoint directory.

`renewer_config_file`
: Location of renewal configuration file.

`no_verify_ssl`
: Disable verification of the ACME server's certificate.

`tls_sni_01_port`
: Port used during tls-sni-01 challenge. This only affects the port Certbot listens on. A conforming ACME server will still attempt to connect on port 443.

`http01_port`
: Port used in the http-01 challenge.This only affects the port Certbot listens on. A conforming ACME server will still attempt to connect on port 80.

interface `certbot.interfaces.`**`IInstaller`**

Extends: *`certbot.interfaces.IPlugin`*

Generic Certbot Installer Interface.

Represents any server that an X509 certificate can be placed.

It is assumed that *`save()`* is the only method that finalizes a checkpoint. This is important to ensure that checkpoints are restored in a consistent manner if requested by the user or in case of an error.

Using *`certbot.reverter.Reverter`* to implement checkpoints, rollback, and recovery can dramatically simplify plugin development.

`get_all_names``()`
: Returns all names that may be authenticated.

Return type `list` of `str`

deploy_cert(*domain*, *cert_path*, *key_path*, *chain_path*, *fullchain_path*)
Deploy certificate.

Parameters

- **domain** (*str*) – domain to deploy certificate file
- **cert_path** (*str*) – absolute path to the certificate file
- **key_path** (*str*) – absolute path to the private key file
- **chain_path** (*str*) – absolute path to the certificate chain file
- **fullchain_path** (*str*) – absolute path to the certificate fullchain file (cert plus chain)

Raises ***PluginError*** – when cert cannot be deployed

enhance(*domain*, *enhancement*, *options=None*)
Perform a configuration enhancement.

Parameters

- **domain** (*str*) – domain for which to provide enhancement
- **enhancement** (*str*) – An enhancement as defined in *ENHANCEMENTS*
- **options** – Flexible options parameter for enhancement. Check documentation of *ENHANCEMENTS* for expected options for each enhancement.

Raises ***PluginError*** – If Enhancement is not supported, or if an error occurs during the enhancement.

supported_enhancements()
Returns a list of supported enhancements.

Returns supported enhancements which should be a subset of *ENHANCEMENTS*

Return type `list` of `str`

get_all_certs_keys()
Retrieve all certs and keys set in configuration.

Returns

tuples with form `[(cert, key, path)]`, where:

- `cert` - str path to certificate file
- `key` - str path to associated key file
- `path` - file path to configuration file

Return type list

save(*title=None*, *temporary=False*)
Saves all changes to the configuration files.

Both title and temporary are needed because a save may be intended to be permanent, but the save is not ready to be a full checkpoint.

It is assumed that at most one checkpoint is finalized by this method. Additionally, if an exception is raised, it is assumed a new checkpoint was not finalized.

Parameters

- **title** (*str*) – The title of the save. If a title is given, the configuration will be saved as a new checkpoint and put in a timestamped directory. `title` has no effect if temporary is true.
- **temporary** (*bool*) – Indicates whether the changes made will be quickly reversed in the future (challenges)

Raises ***PluginError*** – when save is unsuccessful

rollback_checkpoints(*rollback=1*)
Revert `rollback` number of configuration checkpoints.

Raises ***PluginError*** – when configuration cannot be fully reverted

recovery_routine()
Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises ***errors.PluginError*** – If unable to recover the configuration

view_config_changes()
Display all of the LE config changes.

Raises ***PluginError*** – when config changes cannot be parsed

config_test()
Make sure the configuration is valid.

Raises ***MisconfigurationError*** – when the config is not in a usable state

restart()
Restart or refresh the server content.

Raises ***PluginError*** – when server cannot be restarted

interface `certbot.interfaces.`**IDisplay**
Generic display.

notification(*message*, *pause*)
Displays a string message

Parameters

- **message** (*str*) – Message to display
- **pause** (*bool*) – Whether or not the application should pause for confirmation (if available)

menu(*message*, *choices*, *ok_label='OK'*, *cancel_label='Cancel'*, *help_label='‘*, *default=None*, *cli_flag=None*)
Displays a generic menu.

Parameters

- **message** (*str*) – message to display
- **choices** (`list` of `tuple()` or `str`) – choices
- **ok_label** (*str*) – label for OK button
- **cancel_label** (*str*) – label for Cancel button
- **help_label** (*str*) – label for Help button
- **default** (*int*) – default (non-interactive) choice from the menu

- **cli_flag** (*str*) – to automate choice from the menu, eg "–keep"

Returns tuple of (code, index) where code - str display exit code index - int index of the user's selection

Raises ***errors.MissingCommandlineFlag*** – if called in non-interactive mode without a default set

input (*message*, *default=None*, *cli_args=None*)
Accept input from the user.

Parameters **message** (*str*) – message to display to the user

Returns tuple of (code, *input*) where code - str display exit code *input* - str of the user's input

Return type tuple

Raises ***errors.MissingCommandlineFlag*** – if called in non-interactive mode without a default set

yesno (*message*, *yes_label='Yes'*, *no_label='No'*, *default=None*, *cli_args=None*)
Query the user with a yes/no question.

Yes and No label must begin with different letters.

Parameters

- **message** (*str*) – question for the user
- **default** (*str*) – default (non-interactive) choice from the menu
- **cli_flag** (*str*) – to automate choice from the menu, eg "–redirect / –no-redirect"

Returns True for "Yes", False for "No"

Return type bool

Raises ***errors.MissingCommandlineFlag*** – if called in non-interactive mode without a default set

checklist (*message*, *tags*, *default_state*, *default=None*, *cli_args=None*)
Allow for multiple selections from a menu.

Parameters

- **message** (*str*) – message to display to the user
- **tags** (*list*) – where each is of type str len(tags) > 0
- **default_status** (*bool*) – If True, items are in a selected state by default.
- **default** (*str*) – default (non-interactive) state of the checklist
- **cli_flag** (*str*) – to automate choice from the menu, eg "–domains"

Returns tuple of the form (code, list_tags) where code - int display exit code list_tags - list of str tags selected by the user

Return type tuple

Raises ***errors.MissingCommandlineFlag*** – if called in non-interactive mode without a default set

directory_select (*self*, *message*, *default=None*, *cli_flag=None*)
Display a directory selection screen.

Parameters

- **message** (*str*) – prompt to give the user
- **default** – the default value to return, if one exists, when using the NoninteractiveDisplay
- **cli_flag** (*str*) – option used to set this value with the CLI, if one exists, to be included in error messages given by NoninteractiveDisplay

Returns tuple of the form (`code`, `string`) where `code` - int display exit code `string` - input entered by the user

interface `certbot.interfaces.`**`IValidator`**
Configuration validator.

`certificate`(*cert*, *name*, *alt_host=None*, *port=443*)
Verifies the certificate presented at name is cert

Parameters

- **cert** (*OpenSSL.crypto.X509*) – Expected certificate
- **name** (*str*) – Server's domain name
- **alt_host** (*bytes*) – Host to connect to instead of the IP address of host
- **port** (*int*) – Port to connect to

Returns True if the certificate was verified successfully

Return type bool

`redirect`(*name*, *port=80*, *headers=None*)
Verify redirect to HTTPS

Parameters

- **name** (*str*) – Server's domain name
- **port** (*int*) – Port to connect to
- **headers** (*dict*) – HTTP headers to include in request

Returns True if redirect is successfully enabled

Return type bool

`hsts`(*name*)
Verify HSTS header is enabled

Parameters **name** (*str*) – Server's domain name

Returns True if HSTS header is successfully enabled

Return type bool

`ocsp_stapling`(*name*)
Verify ocsp stapling for domain

Parameters **name** (*str*) – Server's domain name

Returns True if ocsp stapling is successfully enabled

Return type bool

interface `certbot.interfaces.`**`IReporter`**
Interface to collect and display information to the user.

HIGH_PRIORITY
Used to denote high priority messages

MEDIUM_PRIORITY
Used to denote medium priority messages

LOW_PRIORITY
Used to denote low priority messages

add_message(*self*, *msg*, *priority*, *on_crash=True*)
Adds msg to the list of messages to be printed.

Parameters

- **msg** (*str*) – Message to be displayed to the user.
- **priority** (*int*) – One of HIGH_PRIORITY, MEDIUM_PRIORITY, or LOW_PRIORITY.
- **on_crash** (*bool*) – Whether or not the message should be printed if the program exits abnormally.

print_messages(*self*)
Prints messages to the user and clears the message queue.

certbot.log

certbot.plugins.common

Plugin common functions.

certbot.plugins.common.**option_namespace**(*name*)
ArgumentParser options namespace (prefix of all options).

certbot.plugins.common.**dest_namespace**(*name*)
ArgumentParser dest namespace (prefix of all destinations).

class certbot.plugins.common.**Plugin**(*config*, *name*)
Bases: object

Generic plugin.

classmethod add_parser_arguments(*add*)
Add plugin arguments to the CLI argument parser.

NOTE: If some of your flags interact with others, you can use cli.report_config_interaction to register this to ensure values are correctly saved/overridable during renewal.

Parameters add (*callable*) – Function that proxies calls to argparse.ArgumentParser.add_argument prepending options with unique plugin name prefix.

classmethod inject_parser_options(*parser*, *name*)
Inject parser options.

See inject_parser_options for docs.

option_namespace
ArgumentParser options namespace (prefix of all options).

`option_name`(*name*)
Option name (include plugin namespace).

`dest_namespace`
ArgumentParser dest namespace (prefix of all destinations).

`dest`(*var*)
Find a destination for given variable `var`.

`conf`(*var*)
Find a configuration value for variable `var`.

class `certbot.plugins.common.`**`Addr`**(*tup*, *ipv6=False*)
Bases: `object`

Represents an virtual host address.

Parameters

- **`addr`** (`str`) – addr part of vhost address
- **`port`** (`str`) – port number or *, or “”

classmethod **`fromstring`**(*str_addr*)
Initialize Addr from string.

`get_addr`()
Return addr part of Addr object.

`get_port`()
Return port.

`get_addr_obj`(*port*)
Return new address object with same addr and new port.

`_normalize_ipv6`(*addr*)
Return IPv6 address in normalized form, helper function

`get_ipv6_exploded`()
Return IPv6 in normalized form

`_explode_ipv6`(*addr*)
Explode IPv6 address for comparison

class `certbot.plugins.common.`**`TLSSNI01`**(*configurator*)
Bases: `object`

Abstract base for TLS-SNI-01 challenge performers

`add_chall`(*achall*, *idx=None*)
Add challenge to TLSSNI01 object to perform at once.

Parameters

- **`achall`** (`KeyAuthorizationAnnotatedChallenge`) – Annotated TLSSNI01 challenge.
- **`idx`** (`int`) – index to challenge in a larger array

`get_cert_path`(*achall*)
Returns standardized name for challenge certificate.

Parameters **`achall`** (`KeyAuthorizationAnnotatedChallenge`) – Annotated tls-sni-01 challenge.

Returns certificate file name

Return type str

get_key_path(*achall*)
Get standardized path to challenge key.

_setup_challenge_cert(*achall*, *cert_key=None*)
Generate and write out challenge certificate.

certbot.plugins.common.**setup_ssl_options**(*config_dir*, *src*, *dest*)
Move the ssl_options into position and return the path.

certbot.plugins.common.**dir_setup**(*test_dir*, *pkg*)
Setup the directories necessary for the configurator.

certbot.plugins.disco

Utilities for plugins discovery and selection.

class certbot.plugins.disco.**PluginEntryPoint**(*entry_point*)
Bases: object

Plugin entry point.

PREFIX_FREE_DISTRIBUTIONS = ['certbot', 'certbot-apache', 'certbot-nginx']
Distributions for which prefix will be omitted.

classmethod entry_point_to_plugin_name(*entry_point*)
Unique plugin name for an `entry_point`

description
Description of the plugin.

description_with_name
Description with name. Handy for UI.

hidden
Should this plugin be hidden from UI?

ifaces(**ifaces_groups*)
Does plugin implements specified interface groups?

initialized
Has the plugin been initialized already?

init(*config=None*)
Memoized plugin inititialization.

verify(*ifaces*)
Verify that the plugin conforms to the specified interfaces.

prepared
Has the plugin been prepared already?

prepare()
Memoized plugin preparation.

misconfigured
Is plugin misconfigured?

problem
Return the Exception raised during plugin setup, or None if all is well

available
Is plugin available, i.e. prepared or misconfigured?

class `certbot.plugins.disco.`**`PluginsRegistry`**(*plugins*)
Bases: `_abcoll.Mapping`

Plugins registry.

classmethod **`find_all`**()
Find plugins using setuptools entry points.

`init`(*config*)
Initialize all plugins in the registry.

`filter`(*pred*)
Filter plugins based on predicate.

`visible`()
Filter plugins based on visibility.

`ifaces`(**ifaces_groups*)
Filter plugins based on interfaces.

`verify`(*ifaces*)
Filter plugins based on verification.

`prepare`()
Prepare all plugins in the registry.

`available`()
Filter plugins based on availability.

`find_init`(*plugin*)
Find an initialized plugin.

This is particularly useful for finding a name for the plugin (although *`IPluginFactory.__call__`* takes `name` as one of the arguments, `IPlugin.name` is not part of the interface):

```
# plugin is an instance providing IPlugin, initialized
# somewhere else in the code
plugin_registry.find_init(plugin).name
```

Returns `None` if `plugin` is not found in the registry.

certbot.plugins.manual

Manual plugin.

class `certbot.plugins.manual.`**`Authenticator`**(**args*, ***kwargs*)
Bases: *`certbot.plugins.common.Plugin`*

Manual Authenticator.

This plugin requires user's manual intervention in setting up a HTTP server for solving http-01 challenges and thus does not need to be run as a privileged process. Alternatively shows instructions on how to use Python's built-in HTTP server.

Todo

Support for `TLSSNI01`.

`CMD_TEMPLATE` **= 'mkdir -p {root}/public_html/{achall.URI_ROOT_PATH}\ncd {root}/public_html\nprintf "%s" {valid**

Command template.

`certbot.plugins.standalone`

Standalone Authenticator.

class `certbot.plugins.standalone.`**`ServerManager`**(*certs*, *http_01_resources*)

Bases: `object`

Standalone servers manager.

Manager for `ACMEServer` and `ACMETLSServer` instances.

`certs` and `http_01_resources` correspond to `acme.crypto_util.SSLSocket.certs` and `acme.crypto_util.SSLSocket.http_01_resources` respectively. All created servers share the same certificates and resources, so if you're running both TLS and non-TLS instances, HTTP01 handlers will serve the same URLs!

class **`_Instance`**(*server*, *thread*)

Bases: `tuple`

`_asdict`()

Return a new OrderedDict which maps field names to their values

classmethod **`_make`**(*iterable*, *new=<built-in method __new__ of type object at 0x906d60>*, *len=<built-in function len>*)

Make a new _Instance object from a sequence or iterable

`_replace`(*_self*, ***kwds*)

Return a new _Instance object replacing specified fields with new values

`server`

Alias for field number 0

`thread`

Alias for field number 1

`ServerManager.`**`run`**(*port*, *challenge_type*)

Run ACME server on specified `port`.

This method is idempotent, i.e. all calls with the same pair of `(port, challenge_type)` will reuse the same server.

Parameters

- **`port`** (*`int`*) – Port to run the server on.
- **`challenge_type`** – Subclass of `acme.challenges.Challenge`, either `acme.challenge.HTTP01` or `acme.challenges.TLSSNI01`.

Returns Server instance.

Return type ACMEServerMixin

`ServerManager.`**`stop`**(*port*)

Stop ACME server running on the specified `port`.

Parameters **`port`** (*`int`*) –

ServerManager.**running**()

Return all running instances.

Once the server is stopped using *stop*, it will not be returned.

Returns Mapping from port to server.

Return type tuple

certbot.plugins.standalone.**supported_challenges_validator**(*data*)

Supported challenges validator for the argparse.

It should be passed as type argument to add_argument.

class certbot.plugins.standalone.**Authenticator**(**args*, ***kwargs*)

Bases: *certbot.plugins.common.Plugin*

Standalone Authenticator.

This authenticator creates its own ephemeral TCP listener on the necessary port in order to respond to incoming tls-sni-01 and http-01 challenges from the certificate authority. Therefore, it does not rely on any existing server program.

supported_challenges

Challenges supported by this plugin.

_verify_ports_are_available(*achalls*)

Confirm the ports are available to solve all achalls.

Parameters **achalls** (*list*) – list of *AnnotatedChallenge*

Raises ***errors.MisconfigurationError*** – if required port is unavailable

perform2(*achalls*)

Perform achallenges without IDisplay interaction.

certbot.plugins.util

Plugin utilities.

certbot.plugins.util.**path_surgery**(*restart_cmd*)

Attempt to perform PATH surgery to find restart_cmd

Mitigates https://github.com/certbot/certbot/issues/1833

Parameters **restart_cmd** (*str*) – the command that is being searched for in the PATH

Returns True if the operation succeeded, False otherwise

certbot.plugins.util.**already_listening**(*port*, *renewer=False*)

Check if a process is already listening on the port.

If so, also tell the user via a display notification.

Warning: On some operating systems, this function can only usefully be run as root.

Parameters **port** (*int*) – The TCP port in question.

Returns True or False.

`certbot.plugins.util.`**`already_listening_socket`**(*port*, *renewer=False*)

Simple socket based check to find out if port is already in use

Parameters **`port`** (*`int`*) – The TCP port in question.

Returns True or False

`certbot.plugins.util.`**`already_listening_psutil`**(*port*, *renewer=False*)

Psutil variant of the open port check

Parameters **`port`** (*`int`*) – The TCP port in question.

Returns True or False.

certbot.plugins.webroot

Webroot plugin.

class `certbot.plugins.webroot.`**`Authenticator`**(**args*, ***kwargs*)

Bases: *`certbot.plugins.common.Plugin`*

Webroot Authenticator.

class `certbot.plugins.webroot.`**`_WebrootMapAction`**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `argparse.Action`

Action class for parsing webroot_map.

class `certbot.plugins.webroot.`**`_WebrootPathAction`**(**args*, ***kwargs*)

Bases: `argparse.Action`

Action class for parsing webroot_path.

`certbot.plugins.webroot.`**`_validate_webroot`**(*webroot_path*)

Validates and returns the absolute path of webroot_path.

Parameters **`webroot_path`** (*`str`*) – path to the webroot directory

Returns absolute path of webroot_path

Return type str

certbot.reporter

Collects and displays information to the user.

class `certbot.reporter.`**`Reporter`**(*config*)

Bases: `object`

Collects and displays information to the user.

Variables **`messages`** (*`queue.PriorityQueue`*) – Messages to be displayed to the user.

`HIGH_PRIORITY` = 0

High priority constant. See *`add_message`*.

`MEDIUM_PRIORITY` = 1

Medium priority constant. See *`add_message`*.

LOW_PRIORITY = 2
Low priority constant. See *add_message*.

_msg_type
alias of `ReporterMsg`

add_message (*msg*, *priority*, *on_crash=True*)
Adds msg to the list of messages to be printed.

Parameters

- **msg** (*str*) – Message to be displayed to the user.
- **priority** (*int*) – One of *HIGH_PRIORITY*, *MEDIUM_PRIORITY*, or *LOW_PRIORITY*.
- **on_crash** (*bool*) – Whether or not the message should be printed if the program exits abnormally.

atexit_print_messages (*pid=None*)
Function to be registered with atexit to print messages.

Parameters pid (*int*) – Process ID

print_messages ()
Prints messages to the user and clears the message queue.

If there is an unhandled exception, only messages for which `on_crash` is `True` are printed.

certbot.reverter

Reverter class saves configuration checkpoints and allows for recovery.

class certbot.reverter.**Reverter** (*config*)
Bases: `object`

Reverter Class - save and revert configuration checkpoints.

This class can be used by the plugins, especially Installers, to undo changes made to the user's system. Modifications to files and commands to do undo actions taken by the plugin should be registered with this class before the action is taken.

Once a change has been registered with this class, there are three states the change can be in. First, the change can be a temporary change. This should be used for changes that will soon be reverted, such as config changes for the purpose of solving a challenge. Changes are added to this state through calls to *add_to_temp_checkpoint()* and reverted when *revert_temporary_config()* or *recovery_routine()* is called.

The second state a change can be in is in progress. These changes are not temporary, however, they also have not been finalized in a checkpoint. A change must become in progress before it can be finalized. Changes are added to this state through calls to *add_to_checkpoint()* and reverted when *recovery_routine()* is called.

The last state a change can be in is finalized in a checkpoint. A change is put into this state by first becoming an in progress change and then calling *finalize_checkpoint()*. Changes in this state can be reverted through calls to *rollback_checkpoints()*.

As a final note, creating new files and registering undo commands are handled specially and use the methods *register_file_creation()* and *register_undo_command()* respectively. Both of these methods can be used to create either temporary or in progress changes.

Note: Consider moving everything over to CSV format.

Parameters **config** (*certbot.interfaces.IConfig*) – Configuration.

revert_temporary_config()
Reload users original configuration files after a temporary save.

This function should reinstall the users original configuration files for all saves with temporary=True

Raises ***ReverterError*** – when unable to revert config

rollback_checkpoints(*rollback=1*)
Revert 'rollback' number of configuration checkpoints.

Parameters **rollback** (*int*) – Number of checkpoints to reverse. A str num will be cast to an integer. So "2" is also acceptable.

Raises ***ReverterError*** – if there is a problem with the input or if the function is unable to correctly revert the configuration checkpoints

view_config_changes(*for_logging=False*, *num=None*)
Displays all saved checkpoints.

All checkpoints are printed by *certbot.interfaces.IDisplay.notification()*.

Todo

Decide on a policy for error handling, OSError IOError...

Raises ***errors.ReverterError*** – If invalid directory structure.

add_to_temp_checkpoint(*save_files*, *save_notes*)
Add files to temporary checkpoint.

Parameters

- **save_files** (*set*) – set of filepaths to save
- **save_notes** (*str*) – notes about changes during the save

add_to_checkpoint(*save_files*, *save_notes*)
Add files to a permanent checkpoint.

Parameters

- **save_files** (*set*) – set of filepaths to save
- **save_notes** (*str*) – notes about changes during the save

_add_to_checkpoint_dir(*cp_dir*, *save_files*, *save_notes*)
Add save files to checkpoint directory.

Parameters

- **cp_dir** (*str*) – Checkpoint directory filepath
- **save_files** (*set*) – set of files to save
- **save_notes** (*str*) – notes about changes made during the save

Raises

- **IOError** – if unable to open cp_dir + FILEPATHS file
- *ReverterError* – if unable to add checkpoint

_read_and_append (*filepath*)
Reads the file lines and returns a file obj.

Read the file returning the lines, and a pointer to the end of the file.

_recover_checkpoint (*cp_dir*)
Recover a specific checkpoint.

Recover a specific checkpoint provided by cp_dir Note: this function does not reload augeas.

Parameters **cp_dir** (*str*) – checkpoint directory file path

Raises *errors.ReverterError* – If unable to recover checkpoint

_run_undo_commands (*filepath*)
Run all commands in a file.

_check_tempfile_saves (*save_files*)
Verify save isn't overwriting any temporary files.

Parameters **save_files** (*set*) – Set of files about to be saved.

Raises *certbot.errors.ReverterError* – when save is attempting to overwrite a temporary file.

register_file_creation (*temporary*, **files*)
Register the creation of all files during certbot execution.

Call this method before writing to the file to make sure that the file will be cleaned up if the program exits unexpectedly. (Before a save occurs)

Parameters

- **temporary** (*bool*) – If the file creation registry is for a temp or permanent save.
- ***files** – file paths (str) to be registered

Raises *certbot.errors.ReverterError* – If call does not contain necessary parameters or if the file creation is unable to be registered.

register_undo_command (*temporary*, *command*)
Register a command to be run to undo actions taken.

Warning: This function does not enforce order of operations in terms of file modification vs. command registration. All undo commands are run first before all normal files are reverted to their previous state. If you need to maintain strict order, you may create checkpoints before and after the the command registration. This function may be improved in the future based on demand.

Parameters

- **temporary** (*bool*) – Whether the command should be saved in the IN_PROGRESS or TEMPORARY checkpoints.
- **command** (*list of str*) – Command to be run.

_get_cp_dir (*temporary*)
Return the proper reverter directory.

recovery_routine()
Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises *errors.ReverterError* – If unable to recover the configuration

_remove_contained_files(*file_list*)
Erase all files contained within file_list.

Parameters **file_list** (*str*) – file containing list of file paths to be deleted

Returns Success

Return type bool

Raises *certbot.errors.ReverterError* – If all files within file_list cannot be removed

finalize_checkpoint(*title*)
Finalize the checkpoint.

Timestamps and permanently saves all changes made through the use of *add_to_checkpoint()* and *register_file_creation()*

Parameters **title** (*str*) – Title describing checkpoint

Raises *certbot.errors.ReverterError* – when the checkpoint is not able to be finalized.

_checkpoint_timestamp()
Determine the timestamp of the checkpoint, enforcing monotonicity.

_timestamp_progress_dir()
Timestamp the checkpoint.

certbot.storage

Renewable certificates storage.

certbot.storage.**config_with_defaults**(*config=None*)
Merge supplied config, if provided, on top of builtin defaults.

certbot.storage.**add_time_interval**(*base_time*, *interval*, *textparser=<parsedatetime.Calendar object>*)
Parse the time specified time interval, and add it to the base_time

The interval can be in the English-language format understood by parsedatetime, e.g., '10 days', '3 weeks', '6 months', '9 hours', or a sequence of such intervals like '6 months 1 week' or '3 days 12 hours'. If an integer is found with no associated unit, it is interpreted by default as a number of days.

Parameters

- **base_time** (*datetime.datetime*) – The time to be added with the interval.
- **interval** (*str*) – The time interval to parse.

Returns The base_time plus the interpretation of the time interval.

Return type datetime.datetime

certbot.storage.**write_renewal_config**(*o_filename*, *n_filename*, *archive_dir*, *target*, *relevant_data*)
Writes a renewal config file with the specified name and values.

Parameters

- **`o_filename`** (*`str`*) – Absolute path to the previous version of config file
- **`n_filename`** (*`str`*) – Absolute path to the new destination of config file
- **`archive_dir`** (*`str`*) – Absolute path to the archive directory
- **`target`** (*`dict`*) – Maps ALL_FOUR to their symlink paths
- **`relevant_data`** (*`dict`*) – Renewal configuration options to save

Returns Configuration object for the new config file

Return type configobj.ConfigObj

`certbot.storage.`**`update_configuration`**(*lineagename*, *archive_dir*, *target*, *cli_config*)

Modifies lineagename's config to contain the specified values.

Parameters

- **`lineagename`** (*`str`*) – Name of the lineage being modified
- **`archive_dir`** (*`str`*) – Absolute path to the archive directory
- **`target`** (*`dict`*) – Maps ALL_FOUR to their symlink paths
- **`cli_config`** (`RenewerConfiguration`) – parsed command line arguments

Returns Configuration object for the updated config file

Return type configobj.ConfigObj

`certbot.storage.`**`get_link_target`**(*link*)

Get an absolute path to the target of link.

Parameters **`link`** (*`str`*) – Path to a symbolic link

Returns Absolute path to the target of link

Return type str

`certbot.storage.`**`_relevant`**(*option*)

Is this option one that could be restored for future renewal purposes? :param str option: the name of the option

Return type bool

`certbot.storage.`**`relevant_values`**(*all_values*)

Return a new dict containing only items relevant for renewal.

Parameters **`all_values`** (*`dict`*) – The original values.

Returns A new dictionary containing items that can be used in renewal.

Rtype dict

class `certbot.storage.`**`RenewableCert`**(*config_filename*, *cli_config*, *update_symlinks=False*)

Bases: `object`

Renewable certificate.

Represents a lineage of certificates that is under the management of Certbot, indicated by the existence of an associated renewal configuration file.

Note that the notion of "current version" for a lineage is maintained on disk in the structure of symbolic links, and is not explicitly stored in any instance variable in this object. The RenewableCert object is able to determine information about the current (or other) version by accessing data on disk, but does not inherently know any of this information except by examining the symbolic links as needed. The instance variables mentioned below point to symlinks that reflect the notion of "current version" of each managed object, and it is these paths that should be used when configuring servers to use the certificate managed in a lineage. These paths are normally within the "live" directory, and their symlink targets – the actual cert files – are normally found within the "archive" directory.

Variables

- **`cert`** (*`str`*) – The path to the symlink representing the current version of the certificate managed by this lineage.
- **`privkey`** (*`str`*) – The path to the symlink representing the current version of the private key managed by this lineage.
- **`chain`** (*`str`*) – The path to the symlink representing the current version of the chain managed by this lineage.
- **`fullchain`** (*`str`*) – The path to the symlink representing the current version of the fullchain (combined chain and cert) managed by this lineage.
- ***`configuration`*** (*`configobj.ConfigObj`*) – The renewal configuration options associated with this lineage, obtained from parsing the renewal configuration file and/or systemwide defaults.

`archive_dir`
Returns the default or specified archive directory

`_check_symlinks``()`
Raises an exception if a symlink doesn't exist

`_update_symlinks``()`
Updates symlinks to use archive_dir

`_consistent``()`
Are the files associated with this lineage self-consistent?

Returns Whether the files stored in connection with this lineage appear to be correct and consistent with one another.

Return type bool

`_fix``()`
Attempt to fix defects or inconsistencies in this lineage.

Todo

Currently unimplemented.

`_previous_symlinks``()`
Returns the kind and path of all symlinks used in recovery.

Returns list of (kind, symlink) tuples

Return type list

`_fix_symlinks``()`
Fixes symlinks in the event of an incomplete version update.

If there is no problem with the current symlinks, this function has no effect.

`current_target``(`*kind*`)`
Returns full path to which the specified item currently points.

Parameters **`kind`** (*`str`*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")

Returns The path to the current version of the specified member.

Return type str or None

`current_version`(*kind*)

Returns numerical version of the specified item.

For example, if kind is "chain" and the current chain link points to a file named "chain7.pem", returns the integer 7.

Parameters **`kind`** (*`str`*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")

Returns the current version of the specified member.

Return type int

`version`(*kind*, *version*)

The filename that corresponds to the specified version and kind.

Warning: The specified version may not exist in this lineage. There is no guarantee that the file path returned by this method actually exists.

Parameters

- **`kind`** (*`str`*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")
- **`version`** (*`int`*) – the desired version

Returns The path to the specified version of the specified member.

Return type str

`available_versions`(*kind*)

Which alternative versions of the specified kind of item exist?

The archive directory where the current version is stored is consulted to obtain the list of alternatives.

Parameters **`kind`** (*`str`*) – the lineage member item (`cert`, `privkey`, `chain`, or `fullchain`)

Returns all of the version numbers that currently exist

Return type `list` of `int`

`newest_available_version`(*kind*)

Newest available version of the specified kind of item?

Parameters **`kind`** (*`str`*) – the lineage member item (`cert`, `privkey`, `chain`, or `fullchain`)

Returns the newest available version of this member

Return type int

`latest_common_version`()

Newest version for which all items are available?

Returns the newest available version for which all members (`cert, ``privkey`, `chain`, and `fullchain`) exist

Return type int

`next_free_version`()

Smallest version newer than all full or partial versions?

Returns the smallest version number that is larger than any version of any item currently stored in this lineage

Return type int

ensure_deployed()

Make sure we've deployed the latest version.

Returns False if a change was needed, True otherwise

Return type bool

May need to recover from rare interrupted / crashed states.

has_pending_deployment()

Is there a later version of all of the managed items?

Returns `True` if there is a complete version of this lineage with a larger version number than the current version, and `False` otherwis

Return type bool

_update_link_to(*kind*, *version*)

Make the specified item point at the specified version.

(Note that this method doesn't verify that the specified version exists.)

Parameters

- **kind** (*str*) – the lineage member item ("cert", "privkey", "chain", or "fullchain")
- **version** (*int*) – the desired version

update_all_links_to(*version*)

Change all member objects to point to the specified version.

Parameters **version** (*int*) – the desired version

names(*version=None*)

What are the subject names of this certificate?

(If no version is specified, use the current version.)

Parameters **version** (*int*) – the desired version number

Returns the subject names

Return type `list` of `str`

Raises ***CertStorageError*** – if could not find cert file.

autodeployment_is_enabled()

Is automatic deployment enabled for this cert?

If autodeploy is not specified, defaults to True.

Returns True if automatic deployment is enabled

Return type bool

should_autodeploy(*interactive=False*)

Should this lineage now automatically deploy a newer version?

This is a policy question and does not only depend on whether there is a newer version of the cert. (This considers whether autodeployment is enabled, whether a relevant newer version exists, and whether the time interval for autodeployment has been reached.)

Parameters **interactive** (*bool*) – set to True to examine the question regardless of whether the renewal configuration allows automated deployment (for interactive use). Default False.

Returns whether the lineage now ought to autodeploy an existing newer cert version

Return type bool

ocsp_revoked(*version=None*)

Is the specified cert version revoked according to OCSP?

Also returns True if the cert version is declared as intended to be revoked according to Let's Encrypt OCSP extensions. (If no version is specified, uses the current version.)

This method is not yet implemented and currently always returns False.

Parameters **version** (*int*) – the desired version number

Returns whether the certificate is or will be revoked

Return type bool

autorenewal_is_enabled()

Is automatic renewal enabled for this cert?

If autorenew is not specified, defaults to True.

Returns True if automatic renewal is enabled

Return type bool

should_autorenew(*interactive=False*)

Should we now try to autorenew the most recent cert version?

This is a policy question and does not only depend on whether the cert is expired. (This considers whether autorenewal is enabled, whether the cert is revoked, and whether the time interval for autorenewal has been reached.)

Note that this examines the numerically most recent cert version, not the currently deployed version.

Parameters **interactive** (*bool*) – set to True to examine the question regardless of whether the renewal configuration allows automated renewal (for interactive use). Default False.

Returns whether an attempt should now be made to autorenew the most current cert version in this lineage

Return type bool

classmethod new_lineage(*lineagename*, *cert*, *privkey*, *chain*, *cli_config*)

Create a new certificate lineage.

Attempts to create a certificate lineage – enrolled for potential future renewal – with the (suggested) lineage name lineagename, and the associated cert, privkey, and chain (the associated fullchain will be created automatically). Optional configurator and renewalparams record the configuration that was originally used to obtain this cert, so that it can be reused later during automated renewal.

Returns a new RenewableCert object referring to the created lineage. (The actual lineage name, as well as all the relevant file paths, will be available within this object.)

Parameters

- **lineagename** (*str*) – the suggested name for this lineage (normally the current cert's first subject DNS name)
- **cert** (*str*) – the initial certificate version in PEM format
- **privkey** (*str*) – the private key in PEM format

- **chain** (*str*) – the certificate chain in PEM format
- **cli_config** (RenewerConfiguration) – parsed command line arguments

Returns the newly-created RenewalCert object

Return type `storage.renewableCert`

save_successor (*prior_version*, *new_cert*, *new_privkey*, *new_chain*, *cli_config*)
Save new cert and chain as a successor of a prior version.

Returns the new version number that was created.

Note: this function does NOT update links to deploy this version

Parameters

- **prior_version** (*int*) – the old version to which this version is regarded as a successor (used to choose a privkey, if the key has not changed, but otherwise this information is not permanently recorded anywhere)
- **new_cert** (*str*) – the new certificate, in PEM format
- **new_privkey** (*str*) – the new private key, in PEM format, or `None`, if the private key has not changed
- **new_chain** (*str*) – the new chain, in PEM format
- **cli_config** (RenewerConfiguration) – parsed command line arguments

Returns the new version number that was created

Return type int

certbot.util

Utilities for all Certbot.

class `certbot.util.`**Key** (*file*, *pem*)
Bases: `tuple`

_asdict ()
Return a new OrderedDict which maps field names to their values

classmethod _make (*iterable*, *new=<built-in method __new__ of type object at 0x906d60>*, *len=<built-in function len>*)
Make a new Key object from a sequence or iterable

_replace (*_self*, ***kwds*)
Return a new Key object replacing specified fields with new values

file
Alias for field number 0

pem
Alias for field number 1

class `certbot.util.`**CSR** (*file*, *data*, *form*)
Bases: `tuple`

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable*, *new=<built-in method __new__ of type object at 0x906d60>*, *len=<built-in function len>*)
Make a new CSR object from a sequence or iterable

_replace(*_self*, ***kwds*)
Return a new CSR object replacing specified fields with new values

data
Alias for field number 1

file
Alias for field number 0

form
Alias for field number 2

certbot.util.**run_script**(*params*)
Run the script with the given params.

Parameters **params** (*list*) – List of parameters to pass to Popen

certbot.util.**exe_exists**(*exe*)
Determine whether path/name refers to an executable.

Parameters **exe** (*str*) – Executable path or name

Returns If exe is a valid executable

Return type bool

certbot.util.**make_or_verify_dir**(*directory*, *mode=493*, *uid=0*, *strict=False*)
Make sure directory exists with proper permissions.

Parameters

- **directory** (*str*) – Path to a directory.
- **mode** (*int*) – Directory mode.
- **uid** (*int*) – Directory owner.
- **strict** (*bool*) – require directory to be owned by current user

Raises

- ***errors.Error*** – if a directory already exists, but has wrong permissions or owner
- **OSError** – if invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

certbot.util.**check_permissions**(*filepath*, *mode*, *uid=0*)
Check file or directory permissions.

Parameters

- **filepath** (*str*) – Path to the tested file (or directory).
- **mode** (*int*) – Expected file mode.
- **uid** (*int*) – Expected file owner.

Returns True if `mode` and `uid` match, False otherwise.

Return type bool

`certbot.util.`**`safe_open`**(*path*, *mode='w'*, *chmod=None*, *buffering=None*)

Safely open a file.

Parameters

- **`path`** (`str`) – Path to a file.
- **`mode`** (`str`) – Same os `mode` for `open`.
- **`chmod`** (`int`) – Same as `mode` for `os.open`, uses Python defaults if `None`.
- **`buffering`** (`int`) – Same as `bufsize` for `os.fdopen`, uses Python defaults if `None`.

`certbot.util.`**`unique_file`**(*path*, *mode=511*)

Safely finds a unique file.

Parameters

- **`path`** (`str`) – path/filename.ext
- **`mode`** (`int`) – File mode

Returns tuple of file object and file name

`certbot.util.`**`unique_lineage_name`**(*path*, *filename*, *mode=511*)

Safely finds a unique file using lineage convention.

Parameters

- **`path`** (`str`) – directory path
- **`filename`** (`str`) – proposed filename
- **`mode`** (`int`) – file mode

Returns tuple of file object and file name (which may be modified from the requested one by appending digits to ensure uniqueness)

Raises **`OSError`** – if writing files fails for an unanticipated reason, such as a full disk or a lack of permission to write to specified location.

`certbot.util.`**`safely_remove`**(*path*)

Remove a file that may not exist.

`certbot.util.`**`get_os_info`**(*filepath='/etc/os-release'*)

Get OS name and version

Parameters **`filepath`** (`str`) – File path of os-release file

Returns (os_name, os_version)

Return type `tuple` of `str`

`certbot.util.`**`get_os_info_ua`**(*filepath='/etc/os-release'*)

Get OS name and version string for User Agent

Parameters **`filepath`** (`str`) – File path of os-release file

Returns os_ua

Return type `str`

`certbot.util.`**`get_systemd_os_info`**(*filepath='/etc/os-release'*)

Parse systemd /etc/os-release for distribution information

Parameters **`filepath`** (`str`) – File path of os-release file

Returns (os_name, os_version)

Return type `tuple` of `str`

`certbot.util.`**`get_systemd_os_like`**(*filepath='/etc/os-release'*)

Get a list of strings that indicate the distribution likeness to other distributions.

Parameters **`filepath`** (*`str`*) – File path of os-release file

Returns List of distribution acronyms

Return type `list` of `str`

`certbot.util.`**`_get_systemd_os_release_var`**(*varname*, *filepath='/etc/os-release'*)

Get single value from systemd /etc/os-release

Parameters

- **`varname`** (*`str`*) – Name of variable to fetch
- **`filepath`** (*`str`*) – File path of os-release file

Returns requested value

Return type `str`

`certbot.util.`**`_normalize_string`**(*orig*)

Helper function for _get_systemd_os_release_var() to remove quotes and whitespaces

`certbot.util.`**`get_python_os_info`**()

Get Operating System type/distribution and major version using python platform module

Returns (os_name, os_version)

Return type `tuple` of `str`

`certbot.util.`**`safe_email`**(*email*)

Scrub email address before using it.

`certbot.util.`**`add_deprecated_argument`**(*add_argument*, *argument_name*, *nargs*)

Adds a deprecated argument with the name argument_name.

Deprecated arguments are not shown in the help. If they are used on the command line, a warning is shown stating that the argument is deprecated and no other action is taken.

Parameters

- **`add_argument`** (*`callable`*) – Function that adds arguments to an argument parser/group.
- **`argument_name`** (*`str`*) – Name of deprecated argument.
- **`nargs`** – Value for nargs when adding the argument to argparse.

`certbot.util.`**`enforce_le_validity`**(*domain*)

Checks that Let's Encrypt will consider domain to be valid.

Parameters **`domain`** (`str` or `unicode`) – FQDN to check

Returns The domain cast to `str`, with ASCII-only contents

Return type str

Raises ***`ConfigurationError`*** – for invalid domains and cases where Let's Encrypt currently will not issue certificates

`certbot.util.`**`enforce_domain_sanity`**(*domain*)

Method which validates domain value and errors out if the requirements are not met.

Parameters **`domain`** (`str` or `unicode`) – Domain to check

Raises `ConfigurationError` – for invalid domains and cases where Let's Encrypt currently will not issue certificates

Returns The domain cast to `str`, with ASCII-only contents

Return type str

`certbot.util.`**`get_strict_version`**`(`*normalized*`)`

Converts a normalized version to a strict version.

Parameters **`normalized`** (*`str`*) – normalized version string

Returns An equivalent strict version

Return type distutils.version.StrictVersion

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX