python$_{s}ecrets Documentation$

Release 23.4.3.dev0+g260677f.d20230420

Dave Dittrich

Apr 20, 2023

Contents

Contents:

CHAPTER 1

psec (python_secrets)

Python command line app for managing groups of secrets (passwords, API keys, etc) and other project variables. Reduces security risks from things like weak default passwords, secrets stored in files in the source code repository directory.

Version: 23.4.2

- Free software: Apache 2.0 License
- Documentation: https://python_secrets.readthedocs.org.

1.1 Features

- Uses the openstack/cliff command line framework for a robust and full-featured CLI. It is easy to add new commands and features!
- Supports a "drop-in" model for defining variables in a modular manner (something like the python-update-dotdee program), supporting simplified bulk setting or generating values of variables as needed.
- Like python-update-dotdee, *psec* produces a single master `.json` file to hold variables defined by the drop-in group description files. That means you can use that file directly to set variables to be used from within other programs like Ansible (e.g. `ansible-playbook playbook.yml -e @"$(psec secrets path)"`)
- Support multiple simultaneous sets of secrets (environments) for flexibility and scalability in multi-environment deployments and to support different use cases or different combinations of secrets.
- Supports changing the storage location of secrets and variables to allow them to be stored on secure mobile media (such as self-encrypting external SSD or Flash drives) or encrypted disk images mounted at run-time to ensure the confidentiality of data at rest.
- List the groups of variables (and how many secrets in each group).

- Describe secrets by their variable name and type (e.g., `password`, `uuid4`, `random_base64`). You can also include a descriptive string to prompt the user for a value, a list of options to choose from (or * for "any value the user enters"), and a list of environment variables to export for other programs to use at run time.
- Allows manual entry of values, setting non-secret variables from a default value, or automatic generation of secrets according to their type.
- Manually set `string` variables based on the output of simple commands. This allows interfacing with external programs for obtaining secrets, such as Vault by Hashicorp.
- Generate unique values for variables, or use a single value per type to simplify use of secrets in access control of services while supporting a "break-glass" process to quickly regenerate secrets when needed.
- Show the variables and their unredacted values (or redacted them to maintain secrecy during demonstrations or in documentation).
- Export the variables (optionally with a specific prefix string) to the environment and run a command that inherits them (e.g., to pass variables to terraform for provisioning cloud instances).
- Output the variables and values in multiple different formats (CSV, JSON, YAML) for use in shell scripts, etc. using `cliff` features.
- Send secrets to other users on demand using GPG encrypted email to protect the secrets while in transit and while at rest in users' email inboxes.
- Makes it easy to store temporary files (e.g., the output from Jinja template rendering) that may contain secrets *outside* of the source repo directory in an environment-specific `tmp/` directory.

Note: Due to the use of the Python `secrets` module, which was introduced in Python 3.6, only Python versions >= 3.6 can be used.

1.2 Limitations

- Secrets are stored in *unencrypted* form in the environments directories. Permissions are set to limit access, but this is not an "encrypt data at rest" solution like Vault by Hashicorp.
- Does not handle secure distributed access for users on remote systems. You must use something like Vault by Hashicorp or libfuse/sshfs for secure (realtime) distributed access.
- Does not handle secure distribution of newly generated secrets out to distributed systems that need them. You will need to use a program like Ansible and related playbooks for pushing out and changing secrets (or for retrieving backups). Look at the D2 Ansible playbooks (https://github.com/davedittrich/ansible-dims-playbooks) for example playbooks for doing these tasks.
- Does not clean up the environment-specific `tmp/` directories. (You need to handle that in code, but at least they are less likely to end up in a Git commit.)

1.3 Usage Concepts

There is a separate **Usage** chapter with individual command documentation. The remainder of this section covers higher level usage concepts necessary to best use the `python_secrets` package in your open source software project.

1.3.1 Directories and files

There are three file system concepts that are important to understand regarding secrets storage:

1. The root *secrets base directory* for secrets storage;
2. The *environment* for organizing a set of secrets and secret group descriptions;
3. The *secrets* file and *group descriptions*.

```
/Users/dittrich/.secrets

0 directories, 0 files
dittrich at ren in ~/git on master▲
$ # you can clone secrets from descriptions in a source repo
dittrich at ren in ~/git on master▲
$ cd /tmp
dittrich at ren in /tmp
$ git clone https://github.com/davedittrich/goSecure.git
Cloning into 'goSecure'...
remote: Counting objects: 1027, done.
remote: Compressing objects: 100% (1[illegible]/191), done.
remote: Total 1027 (delta 176), reus[illegible] (delta 105), pack-reused 71
Receiving objects: 100% (1027/1027), [illegible]| 6.83 MiB/s, done.
Resolving deltas: 100% (535/535), do[illegible]
dittrich at ren in /tmp
$ cd goSecure
dittrich at ren in /tmp/goSecure on master
$ tree secrets
secrets
└── secrets.d
    └── gosecure.yml

1 directory, 1 file
dittrich at ren in /tmp/goSecure on master
$ # secrets des
```

Secrets Base Directory

`psec` expects to store all of files in a directory tree known as a *secrets base directory*. Originally, this was intended to be located in the current user's home directory. Unless you over-ride the name of this directory, it defaults to

`.secrets` on Linux and `secrets` on Windows.

The ability to locate this directory in a different file system path is supported by command line options and an environment variable so you can store files on an exported file share, in a common location for use by a group on a workstation, or to move the contents to an encrypted disk or a different partition with more disk space.

The first time you use ever use `psec`, there will likely be no directory:

```
$ tree ~/.secrets
/Users/dittrich/.secrets [error opening dir]

0 directories, 0 files
```

Note: The secrets base directory may be created automatically for you the first time you create an environment. For more information, see `psec init --help`.

Environments

Environments are sub-directories within the root secrets directory. You can just create the directory structure without any files. You create one environment per set of unique secrets that you need to manage. This could be one for open source *Program A*, one for *Program B*, etc., or it could be one for *development*, one for *testing*, one for *production*, etc. (or any combination).

```
new string variable "gosecure_pi_pubkey" is not defined
new string variable "gosecure_pi_locale" is not defined
new string variable "gosecure_pi_timezone" is not defined
new string variable "gosecure_pi_wifi_country" is not defined
new string variable "gosecure_pi_keyboard_model" is not defined
new string variable "gosecure_pi_keyboard_layout" is not defined
+-----------------------------+----------+--------+----------------------------------+
| Variable                    | Type     | Export | Value                            |
+-----------------------------+----------+--------+----------------------------------+
| gosecure_app_password       | password | None   | preamble unreal tipper theft     |
| gosecure_client_psk         | string   | None   | None                             |
| gosecure_client_ssid        | string   | No     | None                             |
| gosecure_pi_keyboard_layout | string   | No     | None                             |
| gosecure_pi_keyboard_model  | string   | No     | e                                |
| gosecure_pi_locale          | string   | No     | None                             |
| gosecure_pi_password        | password | No     | preamble unreal tipper theft     |
| gosecure_pi_pubkey          | string   | None   | None                             |
| gosecure_pi_timezone        | string   | None   | None                             |
| gosecure_pi_wifi_country    | string   | None   | None                             |
| gosecure_vpn_client_id      | string   | None   | None                             |
| gosecure_vpn_client_psk     | token_hex| None   | e5401676411d7927ad7e9d97cdc4ef3b |
+-----------------------------+----------+--------+----------------------------------+
dittrich at ren in ~/git/goSecure on master*
$ # you can set variables on the command line
dittrich at ren in ~/git/goSecure on master*
$ psec secrets set gosecure_client_psk
```

The command `environments create` creates an environment. Since this program is designed to support multiple environments, a name for the new environment is required. The name of the environment can be provided explicitly, or it can be inferred from the base name of the current working directory:

```
$ pwd
/Users/dittrich/git/python_secrets
$ psec environments create
environment directory /Users/dittrich/.secrets/python_secrets created
$ tree ~/.secrets
/Users/dittrich/.secrets
└── python_secrets
    └── secrets.d

2 directories, 0 files
```

Let's say we want to create empty environments for the three deployments (*development*, *testing*, and *production*). The names can be assigned explicitly by (a) giving an argument on the command line, (b) using the `-e` or `--environment` command line flag, or (c) by setting the environment variable `D2_ENVIRONMENT`:

```
$ psec environments create development
environment directory /Users/dittrich/.secrets/development created
```

(continues on next page)

(continued from previous page)

```
$ psec --environment testing environments create
environment directory /Users/dittrich/.secrets/testing created

$ D2_ENVIRONMENT=production psec environments create
environment directory /Users/dittrich/.secrets/production created

$ tree ~/.secrets
/Users/dittrich/.secrets
├── development
│   └── secrets.d
├── production
│   └── secrets.d
├── python_secrets
│   └── secrets.d
└── testing
    └── secrets.d

8 directories, 0 files
```

If you want to create more than one environment at once, you will have to specify all of the names on the command line as arguments:

```
$ psec environments create development testing production
environment directory /Users/dittrich/.secrets/development created
environment directory /Users/dittrich/.secrets/testing created
environment directory /Users/dittrich/.secrets/production created
```

If you are using one source repository for building multiple deployments, of course you can't rely on the basename of the directory for all deployments. The default environment can be set, shown, or unset, using the `environments default` command.

```
$ psec environments default --help
usage: psec environments default [-h] [--unset-default] [environment]

Manage default environment via file in cwd

positional arguments:
  environment

optional arguments:
  -h, --help       show this help message and exit
  --unset-default  Unset localized environment default
```

If no default is explicitly set, the default that would be applied is returned:

```
$ cd ~/git/python_secrets
$ psec environments default
default environment is "python_secrets"
```

You can get a list of all available environments at any time, including which one would be the default used by sub-commands:

```
$ psec environments list
+-------------+---------+
| Environment | Default |
+-------------+---------+
```

(continues on next page)

(continued from previous page)

```
| development | No      |
| testing     | No      |
| production  | No      |
+-------------+---------+
```

The following shows setting and unsetting the default:

```
$ psec environments default testing
default environment set to "testing"
$ psec environments default
testing
$ psec environments list
+-------------+---------+
| Environment | Default |
+-------------+---------+
| development | No      |
| testing     | Yes     |
| production  | No      |
+-------------+---------+
$ psec environments default --unset-default
default environment unset
```

The environment directories are useable for storing *all* secrets and sensitive files (e.g., backups of certificates, databases, etc.) associated with an environment.

For convenience, there is a command `environments tree` that produces output similar to the Unix `tree` command:

```
$ psec -e d2 environments tree
/Users/dittrich/.secrets/d2
├── backups
│   ├── black.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-06T23:36:58PDT.tgz
│   │   └── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   ├── green.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-06T23:45:49PDT.tgz
│   │   └── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   ├── purple.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   │   ├── trident_2018-01-31T23:38:48PST.tar.bz2
│   │   └── trident_2018-02-04T20:05:33PST.tar.bz2
│   └── red.secretsmgmt.tk
│       ├── letsencrypt_2018-04-06T23:45:49PDT.tgz
│       └── letsencrypt_2018-04-25T16:32:20PDT.tgz
├── dittrich.asc
├── keys
│   └── opendkim
│       └── secretsmgmt.tk
│           ├── 201801.private
│           ├── 201801.txt
│           ├── 201802.private
│           └── 201802.txt
├── secrets.d
│   ├── ca.json
│   ├── consul.json
│   ├── jenkins.json
│   ├── rabbitmq.json
```

(continues on next page)

(continued from previous page)

```
│   ├── trident.json
│   ├── vncserver.json
│   └── zookeper.json
├── secrets.json
└── vault_password.txt
```

To just see the directory structure and not files, add the `--no-files` option:

```
$ psec -e d2 environments tree --no-files
/Users/dittrich/.secrets/d2
├── backups
│   ├── black.secretsmgmt.tk
│   ├── green.secretsmgmt.tk
│   ├── purple.secretsmgmt.tk
│   └── red.secretsmgmt.tk
├── keys
│   └── opendkim
│       └── secretsmgmt.tk
└── secrets.d
```

Secrets and group descriptions

The environment directories just created are all empty. Secrets are stored in a JSON file (`.json`) within the environment's directory, and group descriptions are stored in a drop-in directory with the same base name, but with an extention of `.d` instead of `.json` (following the Linux drop-in configuration style directories used by programs like `rsyslog`, `dnsmasq`, etc.)

The default secrets file name is `secrets.json`, which means the default descriptions directory would be named `secrets.d`.

You can define environment variables to point to the secrets base directory in which a set of different environments can be configured at one time, to define the current environment, and to change the name of the secrets file to something else.

```
$ env | grep ^D2_
D2_SECRETS_BASEDIR=/Users/dittrich/.secrets
D2_ENVIRONMENT=do
```

Each environment is in turn rooted in a directory with the environment's symbolic name (e.g., `do` for DigitalOcean in this example, and `goSecure` for the GitHub davedittrich/goSecure VPN project.)

```
$ tree -L 1 ~/.secrets
/Users/dittrich/.secrets
├── do
└── goSecure

3 directories, 0 files
```

Each set of secrets for a given service or purpose is described in its own file.

```
.
├── secrets.d
│   ├── ca.json
│   ├── consul.json
│   ├── jenkins.json
```

(continues on next page)

(continued from previous page)

```
│   ├── rabbitmq.json
│   ├── trident.json
│   ├── vncserver.json
│   └── zookeper.json
└── secrets.json
```

You can see one of the descriptions files from the template in this repository using `cat tests/secrets.d/myapp.json`:

```
[
  {
    "Variable": "myapp_pi_password",
    "Type": "password",
    "Prompt": "Password for myapp 'pi' user account",
    "Export": "DEMO_pi_password"
  },
  {
    "Variable": "myapp_app_password",
    "Type": "password",
    "Prompt": "Password for myapp web app",
    "Export": "DEMO_app_password"
  },
  {
    "Variable": "myapp_client_psk",
    "Type": "string",
    "Prompt": "Pre-shared key for myapp client WiFi AP",
    "Options": "*",
    "Export": "DEMO_client_psk"
  },
  {
    "Variable": "myapp_client_ssid",
    "Type": "string",
    "Prompt": "SSID for myapp client WiFi AP",
    "Options": "myapp_ssid,*",
    "Export": "DEMO_client_ssid"
  },
  {
    "Variable": "myapp_ondemand_wifi",
    "Type": "boolean",
    "Prompt": "'Connect on demand' when connected to wifi",
    "Options": "true,false",
    "Export": "DEMO_ondemand_wifi"
  },
  {
    "Variable": "myapp_optional_setting",
    "Type": "boolean",
    "Prompt": "Optionally do something",
    "Options": "false,true",
    "Export": "DEMO_options_setting"
  }
]
```

The `psec` program uses the openstack/cliff command line interface framework, which supports multiple output formats. The default format the `table` format, which makes for nice clean output. (Other formats will be described later.)

The groups can be listed using the `groups list` command:

```
$ psec groups list
+---------+-------+
| Group   | Items |
+---------+-------+
| jenkins |     1 |
| myapp   |     4 |
| trident |     2 |
+---------+-------+
```

The variables in one or more groups can be shown with the `groups show` command:

```
$ psec groups show trident myapp
+---------+-----------------------+
| Group   | Variable              |
+---------+-----------------------+
| trident | trident_sysadmin_pass |
| trident | trident_db_pass       |
| myapp   | myapp_app_password    |
| myapp   | myapp_client_psk      |
| myapp   | myapp_client_ssid     |
| myapp   | myapp_ondemand_wifi   |
| myapp   | myapp_pi_password     |
+---------+-----------------------+
```

When integrating a new open source tool or project, you can create a new group and clone its secrets descriptions. This does not copy any values, just the descriptions, allowing the current environment to manage its own values.

```
$ psec groups create newgroup --clone-from ~/git/goSecure/secrets/secrets.d/gosecure.
↪json
created new group "newgroup"
$ psec groups list 2>/dev/null
+----------+-------+
| Group    | Items |
+----------+-------+
| jenkins  |     1 |
| myapp    |     5 |
| newgroup |    12 |
| trident  |     2 |
+----------+-------+
```

1.3.2 Showing Secrets

To examine the secrets, use the `secrets show` command:

```
$ psec secrets show
+-----------------------+----------+----------+-----------------------+
| Variable              | Type     | Value    | Export                |
+-----------------------+----------+----------+-----------------------+
| jenkins_admin_password | password | REDACTED | jenkins_admin_password |
| myapp_app_password    | password | REDACTED | DEMO_app_password      |
| myapp_client_psk      | string   | REDACTED | DEMO_client_ssid       |
| myapp_client_ssid     | string   | REDACTED | DEMO_client_ssid       |
| myapp_ondemand_wifi   | boolean  | REDACTED | DEMO_ondemand_wifi     |
| myapp_pi_password     | password | REDACTED | DEMO_pi_password       |
| trident_db_pass       | password | REDACTED | trident_db_pass        |
```

(continues on next page)

(continued from previous page)

```
| trident_sysadmin_pass   | password | REDACTED | trident_sysadmin_pass   |
+-------------------------+----------+----------+-------------------------+
```

By default, the values of secrets are redacted when output. To show the values in clear text in the terminal output, add the `--no-redact` flag:

```
$ psec secrets show --no-redact
+-------------------------+----------+-------------------------------+------------------
↪------+
| Variable                | Type     | Value                         | Export
↪       |
+-------------------------+----------+-------------------------------+------------------
↪------+
| jenkins_admin_password | password | fetch.outsider.awning.maroon | jenkins_admin_
↪password |
| myapp_app_password     | password | fetch.outsider.awning.maroon | DEMO_app_
↪password       |
| myapp_client_psk       | string   | PSK                           | DEMO_client_psk
↪       |
| myapp_client_ssid      | string   | SSID                          | DEMO_client_ssid
↪       |
| myapp_ondemand_wifi    | boolean  | true                          | DEMO_ondemand_
↪wifi      |
| myapp_pi_password      | password | fetch.outsider.awning.maroon | DEMO_pi_password
↪       |
| trident_db_pass        | password | fetch.outsider.awning.maroon | trident_db_pass
↪       |
| trident_sysadmin_pass  | password | fetch.outsider.awning.maroon | trident_sysadmin_
↪pass   |
+-------------------------+----------+-------------------------------+------------------
↪------+
```

If you don't care about redaction and want to turn it off and save the dozen keystrokes it takes to type '' –no-redact'', you can export the environment variable `D2_NO_REDACT` set to (case-insensitive) "true", "1", or "yes". Anything else leaves the default the same. We'll do this now for later examples.

```
$ export D2_NO_REDACT=true
```

The default is also to show all secrets. If you only want to process a subset of secrets, you have two ways to do this.

1. Specify the variables you want to show on the command line as arguments:

```
$ psec secrets show rabbitmq_default_user_pass rabbitmq_admin_user_pass
+----------------------------+----------+--------------------------------------+
| Variable                   | Type     | Value                                |
+----------------------------+----------+--------------------------------------+
| rabbitmq_default_user_pass | password | handheld.angrily.letdown.frisk       |
| rabbitmq_admin_user_pass   | password | handheld.angrily.letdown.frisk       |
+----------------------------+----------+--------------------------------------+
```

2. Use the `--group` flag and specify the group(s) you want to show as command line arguments:

```
$ psec secrets show --group jenkins trident
+----------------------------+----------+--------------------------------------+
| Variable                   | Type     | Value                                |
+----------------------------+----------+--------------------------------------+
| jenkins_admin_password     | password | handheld.angrily.letdown.frisk       |
```

(continues on next page)

(continued from previous page)

```
| trident_db_pass            | password | handheld.angrily.letdown.frisk       |
| trident_sysadmin_pass      | password | handheld.angrily.letdown.frisk       |
+----------------------------+----------+--------------------------------------+
```

1.3.3 Describing Secrets and Secret Types

To describe the secrets in the select environment, use the `secrets describe` command:

```
$ psec secrets describe
+----------------------------+----------+-----------------------------------------+
| Variable                   | Type     | Prompt                                  |
+----------------------------+----------+-----------------------------------------+
| google_oauth_client_id     | string   | Google OAuth2 client id                 |
| google_oauth_client_secret | string   | Google OAuth2 client secret             |
| google_oauth_refresh_token | string   | Google OAuth2 refresh token             |
| google_oauth_username      | None     | google_oauth_username                   |
| jenkins_admin_password     | password | Password for Jenkins "admin" account    |
| myapp_app_password         | password | Password for myapp web app              |
| myapp_client_psk           | string   | Pre-shared key for myapp client WiFi AP |
| myapp_client_ssid          | string   | SSID for myapp client WiFi AP           |
| myapp_ondemand_wifi        | boolean  | "Connect on demand" when connected to wifi |
| myapp_pi_password          | password | Password for myapp "pi" user account    |
| trident_db_pass            | password | Password for Trident postgres database  |
| trident_sysadmin_pass      | password | Password for Trident sysadmin account   |
+----------------------------+----------+-----------------------------------------+
$ psec secrets describe --group trident
+----------------------+----------+--------------------------------------+
| Variable             | Type     | Prompt                               |
+----------------------+----------+--------------------------------------+
| trident_db_pass      | password | Password for Trident postgres database |
| trident_sysadmin_pass | password | Password for Trident sysadmin account |
+----------------------+----------+--------------------------------------+
```

To get a description of the available secret types, add the `--types` flag.

```
$ psec secrets describe --types
+-----------------+---------------------------------+
| Type            | Description                     |
+-----------------+---------------------------------+
| password        | Simple (xkcd) password string   |
| string          | Simple string                   |
| boolean         | Boolean ("true"/"false")        |
| crypt_6         | crypt() SHA512 ("$6$")          |
| token_hex       | Hexadecimal token               |
| token_urlsafe   | URL-safe token                  |
| sha256_digest   | DIGEST-SHA256 (user:pass) digest |
| uuid4           | UUID4 token                     |
| random_base64   | Random BASE64 token             |
+-----------------+---------------------------------+
```

Note: The type `string` is for secrets that are managed by another entity that you must obtain and use to access some remote service (e.g., the pre-shared key for someone's WiFi network, or an API key for accessing a cloud service provider's platform). All other types are structured secret types that you generate for configuring services.

1.3.4 Generating and Setting variables

Secrets are generated using the `secrets generate` command and are set manually using the `secrets set` command.

```
$ psec help secrets generate
usage: psec secrets generate [-h] [-U] [args [args ...]]

Generate values for secrets

positional arguments:
  args

optional arguments:
  -h, --help    show this help message and exit
  -U, --unique  Generate unique values for each type of secret (default:
                False)

..
```

```
$ psec secrets set --help
usage: psec secrets set [-h] [--undefined] [args [args ...]]

Set values manually for secrets

positional arguments:
  args

optional arguments:
  -h, --help   show this help message and exit
  --undefined  Set values for undefined variables (default: False)
```

To regenerate all of the non-string secrets at once, using the same value for each type of secret to simplify things, use the `secrets generate` command:

```
$ psec secrets generate
$ psec secrets show --column Variable --column Value
+---------------------------+-----------------------------------------------+
| Variable                  | Value                                         |
+---------------------------+-----------------------------------------------+
| trident_db_pass           | gargle.earlobe.eggplant.kissable              |
| ca_rootca_password        | gargle.earlobe.eggplant.kissable              |
| consul_key                | HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY=  |
| jenkins_admin_password    | gargle.earlobe.eggplant.kissable              |
| rabbitmq_default_user_pass| gargle.earlobe.eggplant.kissable              |
| rabbitmq_admin_user_pass  | gargle.earlobe.eggplant.kissable              |
| trident_sysadmin_pass     | gargle.earlobe.eggplant.kissable              |
| vncserver_password        | gargle.earlobe.eggplant.kissable              |
| zookeeper_uuid4           | 769a77ad-b06f-4018-857e-23f970c777c2          |
+---------------------------+-----------------------------------------------+
```

You can set one or more variables manually using `secrets set` and specifying the variable and value in the form `variable=value`:

```
$ psec secrets set trident_db_pass="rural coffee purple sedan"
$ psec secrets show --column Variable --column Value
+---------------------------+-----------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Variable                    | Value                                          |
+-----------------------------+------------------------------------------------+
| trident_db_pass             | rural coffee purple sedan                      |
| ca_rootca_password          | gargle.earlobe.eggplant.kissable               |
| consul_key                  | HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY=   |
| jenkins_admin_password      | gargle.earlobe.eggplant.kissable               |
| rabbitmq_default_user_pass  | gargle.earlobe.eggplant.kissable               |
| rabbitmq_admin_user_pass    | gargle.earlobe.eggplant.kissable               |
| trident_sysadmin_pass       | gargle.earlobe.eggplant.kissable               |
| vncserver_password          | gargle.earlobe.eggplant.kissable               |
| zookeeper_uuid4             | 769a77ad-b06f-4018-857e-23f970c777c2           |
+-----------------------------+------------------------------------------------+
```

Caution: Note in the example above that the command argument is `trident_db_pass="rural coffee purple sedan"` and not `trident_db_pass='rural coffee purple sedan'`. When using the `variable=value` form of the `secrets set` command with a value that contains spaces, you **must** quote the value with the double-quote character (`"`) as opposed to the single-quote (apostrophe, or `'`) character. The Bash shell (and possibly other shells) will not properly parse the command line and the resulting `sys.argv` argument vector will be incorrectly set as seen here:

```
_sys.argv[1:] = {list} <class 'list'>: ['--debug', 'secrets', 'set', 'trident_db_
↪password=rural coffee purple sedan']
 0 = {str} '--debug'
 1 = {str} 'secrets'
 2 = {str} 'set'
 3 = {str} 'trident_db_password=rural coffee purple sedan'
 __len__ = {int} 4


_sys.argv[1:] = {list} <class 'list'>: ['--debug', 'secrets', 'set', "trident_db_
↪password='rural", 'coffee', 'purple', "sedan'"]
 0 = {str} '--debug'
 1 = {str} 'secrets'
 2 = {str} 'set'
 3 = {str} 'trident_db_password=\\'rural'
 4 = {str} 'coffee'
 5 = {str} 'purple'
 6 = {str} 'sedan\\''
 __len__ = {int} 7
```

Or you can generate one or more variables in a similar manner by adding them to the command line as arguments to `secrets generate`:

```
$ psec secrets generate rabbitmq_default_user_pass rabbitmq_admin_user_pass
$ psec secrets show --column Variable --column Value
+-----------------------------+------------------------------------------------+
| Variable                    | Value                                          |
+-----------------------------+------------------------------------------------+
| trident_db_pass             | rural.coffee.purple.sedan                      |
| ca_rootca_password          | gargle.earlobe.eggplant.kissable               |
| consul_key                  | HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY=   |
| jenkins_admin_password      | gargle.earlobe.eggplant.kissable               |
| rabbitmq_default_user_pass  | embezzle.xerox.excess.skydiver                 |
| rabbitmq_admin_user_pass    | embezzle.xerox.excess.skydiver                 |
```

(continues on next page)

(continued from previous page)

```
| trident_sysadmin_pass      | gargle.earlobe.eggplant.kissable              |
| vncserver_password         | gargle.earlobe.eggplant.kissable              |
| zookeeper_uuid4            | 769a77ad-b06f-4018-857e-23f970c777c2          |
+----------------------------+-----------------------------------------------+
```

A set of secrets for an open source project can be bootstrapped using the following steps:

1. Create a template secrets environment directory that contains just the secrets definitions. This example uses the template found in the davedittrich/goSecure repository (directory https://github.com/davedittrich/goSecure/tree/master/secrets).
2. Use this template to clone a secrets environment, which will initially be empty:

```
$ psec environments create test --clone-from ~/git/goSecure/secrets
new password variable "gosecure_app_password" is unset
new string variable "gosecure_client_ssid" is unset
new string variable "gosecure_client_ssid" is unset
new string variable "gosecure_client_psk" is unset
new password variable "gosecure_pi_password" is unset
new string variable "gosecure_pi_pubkey" is unset
environment directory /Users/dittrich/.secrets/test created
```

Note: The warnings about undefined new variables are presented on the standard error file handle (a.k.a., `&2`). You get rid of them on the console by redirecting `stderr` to `/dev/null` or a file:

```
$ psec environments create test --clone-from ~/git/goSecure/secrets 2>/dev/null
environment directory /Users/dittrich/.secrets/test created
```

```
$ psec -e test secrets show --no-redact --fit-width
+-----------------------+----------+-------+
| Variable              | Type     | Value |
+-----------------------+----------+-------+
| gosecure_app_password | password | None  |
| gosecure_client_ssid  | string   | None  |
| gosecure_client_psk   | string   | None  |
| gosecure_pi_password  | password | None  |
| gosecure_pi_pubkey    | string   | None  |
+-----------------------+----------+-------+
```

3. First, generate all secrets whose type is not `string`:

```
$ psec -e test secrets generate
new password variable "gosecure_app_password" is unset
new string variable "gosecure_client_ssid" is unset
new string variable "gosecure_client_ssid" is unset
new string variable "gosecure_client_psk" is unset
new password variable "gosecure_pi_password" is unset
new string variable "gosecure_pi_pubkey" is unset

$ psec -e test secrets show --no-redact --fit-width
+-----------------------+----------+------------------------------+
| Variable              | Type     | Value                        |
+-----------------------+----------+------------------------------+
| gosecure_app_password | password | brunt.outclass.alike.turbine |
```

(continues on next page)

(continued from previous page)

```
| gosecure_client_psk    | string   | None                           |
| gosecure_client_ssid   | string   | None                           |
| gosecure_pi_password   | password | brunt.outclass.alike.turbine |
| gosecure_pi_pubkey     | string   | None                           |
+------------------------+----------+--------------------------------+
```

4. Finally, manually set the remaining `string` type variables:

```
$ psec -e test secrets set --undefined
new string variable "gosecure_client_psk" is unset
new string variable "gosecure_client_ssid" is unset
new string variable "gosecure_pi_pubkey" is unset
Pre-shared key for goSecure client WiFi AP? [None]: atjhK5AlsQMw3Zh
SSID for goSecure client WiFi AP? [None]: YourWiFiSSID
SSH public key for accessing "pi" account? [None]: @~/.ssh/new_rsa.pub

$ psec -e test secrets show --no-redact --fit-width
+------------------------+----------+-----------------------------------------------
↪-----------------------------------------+
| Variable               | Type     | Value                                        ␣
↪                                        |
+------------------------+----------+-----------------------------------------------
↪-----------------------------------------+
| gosecure_app_password | password | brunt.outclass.alike.turbine                 ␣
↪                                        |
| gosecure_client_psk    | string   | atjhK5AlsQMw3Zh
| gosecure_client_ssid   | string   | YourWiFiSSID                                 ␣
↪                                        |
| gosecure_pi_password   | password | brunt.outclass.alike.turbine                 ␣
↪                                        |
| gosecure_pi_pubkey     | string   | ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQC+qUIucrPvRkTmY0tgxr9ac/
↪VtBUHhYHfOdDVpU99AcryLMWiU |
|                        |          | uQ2/
↪NVikfOfPo5mt9YTQyqRbeBzKlNgbHnsxh0AZatjhK5AlsQMw3ZhZUcLYZbt7szuQy8ineN0potlCJoVaMSOb␣
↪|
|                        |          |␣
↪9htf9gAPvzwxUnHxg35jPCzAXYAi3Erc6y338+CL0XxQvCogXOA+MwH7wZGgdT3WpupLG/7HAr/
↪3KJEQQk1FlS2m |
|                        |          |␣
↪Rd+WuewnLbKkqBP21N+48ccq6XhEhAmlzzr9SENw5DMmrvMAYIYkoTwUeD3Qx4YebjFkCxZw+w7AafEFn0Kz6vCX␣
↪|
|                        |          | 4mp/6ZF/Ko+o04HM2sVr6wtCu2dB␣
↪dittrich@localhost                                    |
+------------------------+----------+-----------------------------------------------
↪-----------------------------------------+
```

Note: If you don't want to see the warnings about new variables that are not defined, simply add the `-q` flag.

```
$ psec -q secrets generate
$ psec -q secrets set --undefined
Pre-shared key for goSecure client WiFi AP? [None]:
```

You are now ready to compile your software, or build your project!

There is also a mechanism to run simple commands (i.e., basic arguments with no special inline command substitution or variable expansion features of shells like `bash`) and use the resulting output as the value.

For this example, let's assume an environment that requires a CIDR notation address for ingres access control (e.g., when using Amazon Web Services to allow control of instances from your remote laptop).

```
$ psec -e xgt secrets set aws_cidr_allowed=""
$ psec -e secrets show --no-redact aws_cidr_allowed
+------------------+--------+-------+
| Variable         | Type   | Value |
+------------------+--------+-------+
| aws_cidr_allowed | string |       |
+------------------+--------+-------+
```

The `psec` program has a utility feature that will return the current routable IP source address as an IP address, or using CIDR notation. The variable can be set in one of two ways:

1. Via (non-interactive) inline command subtitution from the terminal shell:

   ```
   $ psec -e xgt secrets set aws_cidr_allowed="$(psec utils myip --cidr)"
   ```

2. Interactively when prompted using simple command line form:

   ```
   $ psec -e xgt secrets set aws_cidr_allowed
   aws_cidr_allowed? []: !psec utils myip --cidr
   ```

The variable now contains the output of the specified program:

```
$ psec secrets show --no-redact aws_cidr_allowed
+------------------+--------+------------------+
| Variable         | Type   | Value            |
+------------------+--------+------------------+
| aws_cidr_allowed | string | 93.184.216.34/32 |
+------------------+--------+------------------+
```

Note: If you work from behind a static NAT firewall, this IP address will likely not change very often (if at all). If you are using a mobile device that is assigned differing DHCP addresses depending on location, the IP address may change fairly regularly and the initial AWS Security Group setting will begin to block access to your cloud instances. Programs like `terraform` can refresh their state, allowing you to simply reset the variable used to create the Security Group and re-apply the plan to regenerate the AWS Security Group and re-enable your remote access.

1.3.5 Sharing secrets

The `psec` program has a mechanism for sharing secrets with others using GPG encrypted email messages for securing secrets in transit and at rest in users' inboxes. Email is sent using Google's OAuth2 authenticated SMTP services.

Note: The Electronic Frontier Foundation (EFF) has a Surveillance Self-Defense Guide that includes guides on How to Use PGP for Linux and other operating systems. Follow their instructions if you are new to PGP/GPG.

The command is `secrets send`.

```
$ psec secrets send --help
usage: psec secrets send [-h] [-T] [--test-smtp] [-H SMTP_HOST]
                         [-U SMTP_USERNAME] [-F SMTP_SENDER] [-S SMTP_SUBJECT]
                         [args [args ...]]

Send secrets using GPG encrypted email. Arguments are USERNAME@EMAIL.ADDRESS
and/or VARIABLE references.

positional arguments:
  args

optional arguments:
  -h, --help            show this help message and exit
  -T, --refresh-token   Refresh Google API Oauth2 token and exit (default:
                        False)
  --test-smtp           Test Oauth2 SMTP authentication and exit (default:
                        False)
  -H SMTP_HOST, --smtp-host SMTP_HOST
                        SMTP host (default: localhost)
  -U SMTP_USERNAME, --smtp-username SMTP_USERNAME
                        SMTP authentication username (default: None)
  -F SMTP_SENDER, --from SMTP_SENDER
                        Sender address (default: 'noreply@nowhere')
  -S SMTP_SUBJECT, --subject SMTP_SUBJECT
                        Subject line (default: 'For Your Information')
```

Any arguments (`args`) that contain the `@` symbol are assumed to be email addresses while the rest are assumed to be the names of secrets variables to be sent.

All recipients must have GPG public keys in your keyring. An exception is thrown if no GPG key is associated with the recipient(s) email addresses.

```
$ psec secrets send dittrich@u.washington.edu myapp_app_password
Setting homedir to '/Users/dittrich/.gnupg'

Initialised settings:
binary: /usr/local/bin/gpg
binary version: 1.4.11\ncfg:pubkey:1;2;3;16;17\ncfg:cipher:2;3;4;7;8;9;10;11;12;
 ↪13\ncfg:ciphername:3DES;CAST5;BLOWFISH;AES;AES192;AES256;TWOFISH;CAMELLIA128;
 ↪CAMELLIA192;CAMELLIA256\ncfg:digest:1;2;3;8;9;10;11\ncfg:digestname:MD5;SHA1;
 ↪RIPEMD160;SHA256;SHA384;SHA512;SHA224\ncfg:compress:0;1;2;3\n'
homedir: /Users/dittrich/.gnupg
ignore_homedir_permissions: False
keyring: /Users/dittrich/.gnupg/pubring.gpg
secring: /Users/dittrich/.gnupg/secring.gpg
default_preference_list: SHA512 SHA384 SHA256 AES256 CAMELLIA256 TWOFISH AES192 ZLIB␣
 ↪ZIP Uncompressed
keyserver: hkp://wwwkeys.pgp.net
options: None
verbose: False
use_agent: False

Creating the trustdb is only available with GnuPG>=2.x
sent encrypted secrets to dittrich@u.washington.edu
```

Use `-q` to produce no extraneous output.

```
$ psec -q secrets send dittrich@u.washington.edu myapp_app_password
```

The resulting email looks like this:

```
Message-ID: <5bac64ce.1c69fb81.b136e.45ae@mx.google.com>
Date: Wed, 26 Sep 2018 22:04:14 -0700 (PDT)
From: dave.dittrich@gmail.com
X-Google-Original-From: noreply@nowhere
Content-Type: multipart/related; boundary="===============6413073026511107073=="
MIME-Version: 1.0
Subject: For Your Information
To: dittrich@u.washington.edu

This is a multi-part message in MIME format.
--===============6413073026511107073==
Content-Type: multipart/alternative; boundary="===============2830935289665347054=="
MIME-Version: 1.0

--===============2830935289665347054==
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: base64

LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tCgpoUUlXQStSZlhnK3dLTGJlRUFnZlFNcjZYb0lT
cS9BaTlMbEVpZTFTejd5ckEzUmN4SWdjb01XTUNSM3JBaXBHCjF0TTJoZkpxRGJZOThSOEVST01F
aVltSzR2aVJ4ZjgrSU54NU54SUJPbFh1T1JQTy82NElUKzdrVSt5aDZGV00KNU1MK0Jkb21sQzNF
eC9pd3hwbTJ1R2FPczFpcU9DaDIxbTd5RnJWYkNVSW5NN1ZiMTEwck41aXNOZ3BFdndrQgpaZHhp
alJqazdtYVl1eFNkc2c3Y2RVQ29uSmdBR214QU0vZkFzOTREcHNrYkwzMFpqZE1iRHlMbUk4NWp2
QU45CjU3KzAxLzM1MEMyN1hrbEUxdEZudWNlRkRqZ04zeEd4K2Zud0pqdkFpNUpaVHltanRkQi9r
dUZUMlJTTmJJTlAKMWRZdHp4WGxNeVd0SVphNDVYcHdNenZ1TkFTbEJtbENjQXk4YlluSEJmeFRy
SGdJSUlCMlZNY1N6dmdjR3BtVApkYzZqaDVOeEV1bWljOWdXMmplSnFqRHRtdW9Ib3dxZldZb2xX
bGlXUTMrNDNzeVkrdHFlMGgvWEwzS2ZxSTMrClZzWWdyQmpGd0hnem1INEthMWxucXdUZkMzZTJ3
cUI4Uk5hcllqcXAzbHFQOVBhMHdzSVVWMHVYN2dhL01kVWcKdHNRSktPWWJRTnlXVTFLZEZWNHl4
Ynp1TWVlQ3ltMmxMbXJwVks5T3hCV04vbCtXMjRsWmhkck9TcGFJQnpNdgpnc1p3VWVuVzBXR054
bklwUGhoSWRuVE40ZlNscE5JVDhMcmJYeUhoY2ZVS2lsUDNpeEVPRS9Lc25QUFJNTURFCk9SY0xT
Z3FMMTB4b0toMnNzZTNxNG5RaHZkZW5IVVVxVjJ0WW1UVmRCNVl3cTN1MFdtY3BGSGU2NnBZeTBB
VSsKdzRjb2JVM2crQWtJMHBNQnllRzZYaWV4VzF1UzRLVVVnaFlhWVlYQ2dnazJZNEpZT05QSDJJ
NlIydmxuNjFsVApZdm1tR0NNamw3cC9pTnE2RWJpbndoMnNsbkpLMHd3S1BIbVBPUjJvRjdWREN0
dE9idHA0cEZUWTNHalByc0dRCkNDT3dYR2hCSFVQRnY2c3R4NEdtUi9GUWpBRWxxaEpjQWtTbDFz
WWhsUFRhSmEyVGgyNG81L1lPUmxRaHhhRUgKUEFrNFgzcGVCMk9UVjRNR2RCOD0KPTc0aXEKLS0t
LS1FTkQgUEdQIE1FU1NBR0UtLS0tLQo=

--===============2830935289665347054==
Content-Type: text/html; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: base64

VGhlIGZvbGxvd2luZyBzZWNyZXQgaXMgYmVpbmcgc2hhcmVkIHdpdGggeW91OgoKbXlhcHBfYXBw
X3Bhc3N3b3JkPWJydW50IG91dGNsYXNzIGFsaWtlIHR1cmJpbmU=

--===============2830935289665347054==--

--===============6413073026511107073==--
```

Decrypted, it looks like this:

```
Date: Wed, 26 Sep 2018 22:04:14 -0700 (PDT)
```

(continues on next page)

(continued from previous page)

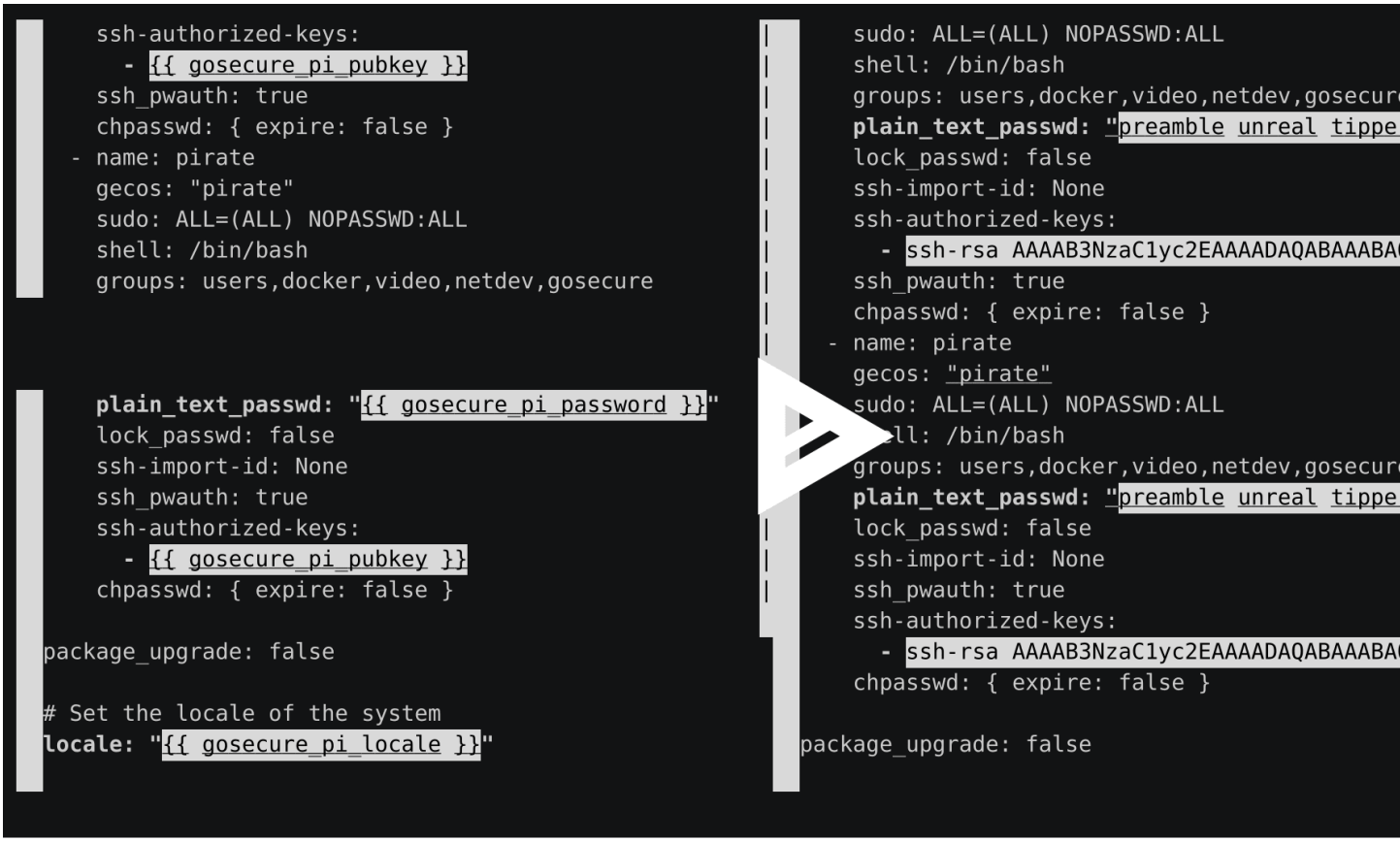
```
From: dave.dittrich@gmail.com
Subject: For Your Information
To: dittrich@u.washington.edu

The following secret is being shared with you:

myapp_app_password=brunt.outclass.alike.turbine

--
Sent using psec version 23.4.2
https://pypi.org/project/python-secrets/
https://github.com/davedittrich/python_secrets
```

A group of secrets required for Google's OAuth 2.0 Mechanism is provided and must be set according to Google's instructions. See also:

- https://github.com/google/gmail-oauth2-tools/wiki/OAuth2DotPyRunThrough
- http://blog.macuyiko.com/post/2016/how-to-send-html-mails-with-oauth2-and-gmail-in-python.html
- https://developers.google.com/api-client-library/python/guide/aaa_oauth
- https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py
- https://developers.google.com/identity/protocols/OAuth2

```
$ psec groups show oauth
+-------+---------------------------+
| Group | Variable                  |
+-------+---------------------------+
| oauth | google_oauth_client_id    |
| oauth | google_oauth_client_secret |
| oauth | google_oauth_refresh_token |
+-------+---------------------------+
```

1.3.6 Processing templates



1.3.7 Outputting structured information for use in other scripts

Once secrets are created and stored, they will eventually need to be accessed in order to use them in program execution. This can be done by passing the `.json` secrets file itself to a program, or by outputting the variables in other formats like CSV, JSON, or as environment type variables.

Passing the secrets file by path

One way to do this is to take advantage of command line options like Ansible's `--extra-vars` and passing it a path to the `.json` secrets file. (See Passing Variables On The Command Line). Here is how to do it.

Let's assume we want to use `consul_key` variable to configure Consul using Ansible. Here is the variable as stored:

```
$ psec secrets show consul_key
+------------+-----------+-----------------------------------------------+
| Variable   | Type      | Value                                         |
+------------+-----------+-----------------------------------------------+
| consul_key | token_hex | HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY= |
+------------+-----------+-----------------------------------------------+
```

Using Ansible's `debug` module, we can verify that this variable is not set by any previously loaded Ansible inventory:

```
$ ansible -i localhost, -m debug -a 'var=consul_key' localhost
localhost | SUCCESS => {
    "consul_key": "VARIABLE IS NOT DEFINED!"
}
```

In order for Ansible to set the `consul_key` variable outside of any pre-defined inventory files, we need to pass a file path to the `--extra-vars` option. The path can be obtained using the `psec secrets path` command:

```
$ psec secrets path
/Users/dittrich/.secrets/python_secrets/secrets.json
```

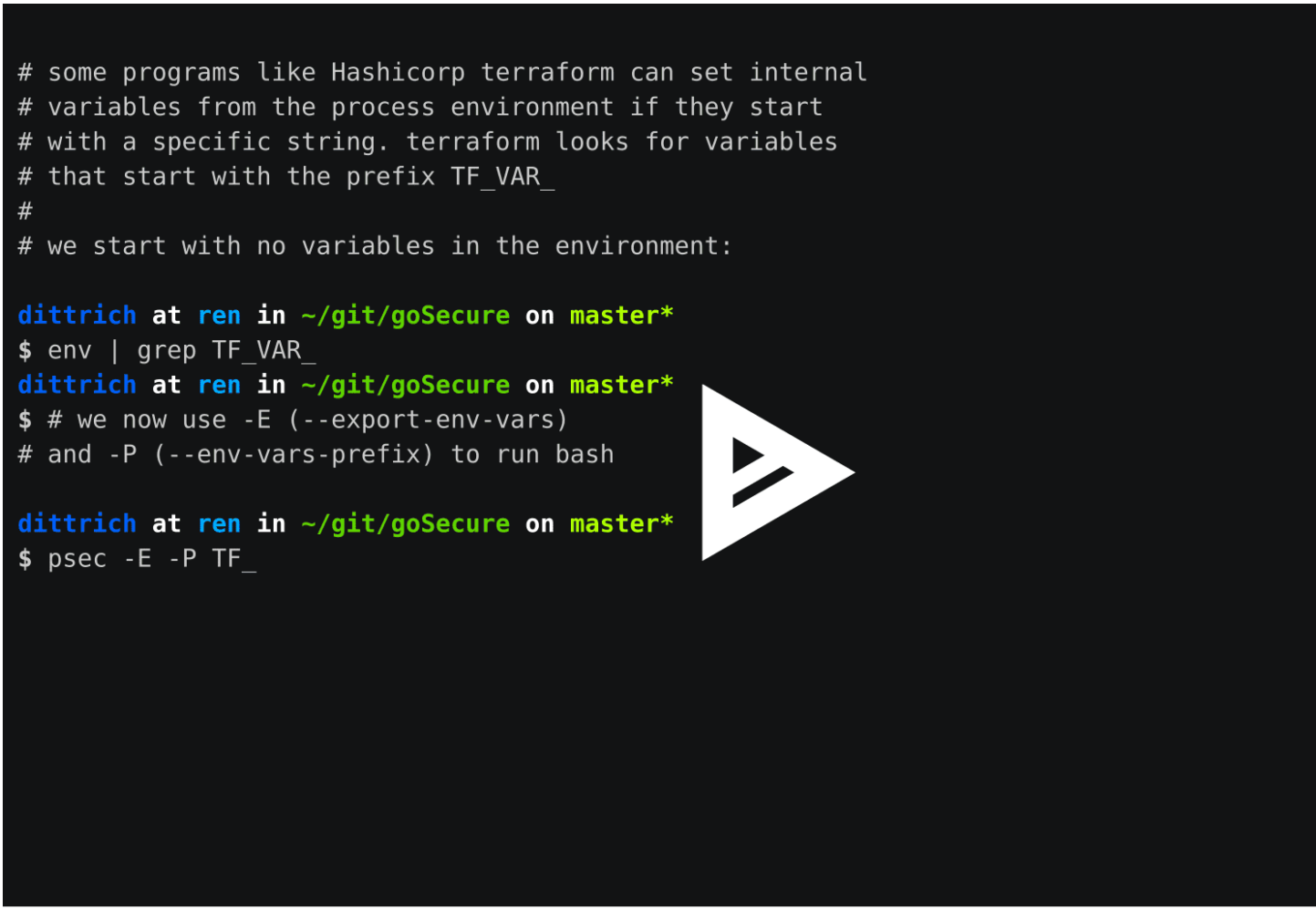
It is possible to run this command in an in-line command expansion operation in Bash. Ansible expects the file path passed to `-extra-vars` to start with an `@` character, so the command line to use would look like this:

```
$ ansible -i localhost, -e @"$(psec secrets path)" -m debug -a 'var=consul_key' ␣
↪localhost
localhost | SUCCESS => {
    "consul_key": "HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
}
```

Ansible now has the value and can use it in templating configuration files, or so forth.

Other programs like Hashicorp terraform look for environment variables that begin with `TF_VAR_` and use them to set `terraform` variables for use in modules. To prove we are running in a sub-shell, we will first change the shell prompt.

```
$ PS1="test> "
test> psec -e test --export-env-vars --env-var-prefix="TEST_" run bash
$ env | grep '^TEST_'
TEST_gosecure_pi_pubkey=ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC+qUIucrPvRkTmY0tgxr9ac/
↪VtBUHhYHfOdDVpU99AcryLMWiU [...]
TEST_gosecure_client_psk=atjhK5AlsQMw3Zh
TEST_gosecure_client_ssid=YourWiFiSSID
TEST_gosecure_pi_password=brunt.outclass.alike.turbine
TEST_gosecure_app_password=brunt.outclass.alike.turbine
$ exit
test>
```



1.4 Operational Security

As noted in the Limitations section above, secrets are stored in plaintext plaintext form (i.e., they are *not* encrypted) in files. Those files are in turn stored in a directory in the file system, subject to Linux file ownership and permission access controls.

The default location for storing these files is in an *environment directory* in a subdirectory of the user's home directory whose name starts with a period character (a.k.a., a *dot*). Files (or directories) whose name starts with a period are known as *dot files*, or *hidden files* because the *ls* command does not show it unless you use the *-a* flag.

The secrets environment directories can also be used to store other files besides secrets. One such use case is storing JSON Web Tokens (JWTs) used as bearer tokens by protocols like Google's OAuth 2.0 Mechanism for securing access to web services and APIs. While this improves security in terms of remote access, is not not without its own risks (including the JWT file being stored in the file system for an indefinite period of time).

- JSON Web Tokens (JWT) are Dangerous for User Sessions—Here's a Solution, by Raja Rao, June 24, 2021
- Stop Using JSON Web Tokens For Authentication. Use Stateful Sessions Instead, by Francisco Sainz, April 4, 2022
- What's the Secure Way to Store JWT?, by Yang Liu, July 23, 2020

Besides JWTs, other use cases for storing sensitive files within *psec* environments include backups of database contents, Let's Encrypt certificates, SSH keys, or other secrets necessary for ensuring cloud instances can be destroyed and recreated without losing state or requiring regeneration (and redistribution or revalidation) of secrets.

The output of *init –help* mentions this risk and offers a way to mitigate some of the risk by locating the secrets storage base directory within a directory that is stored on an encrypted USB-connected disk device or encrypted disk image, or a removable device or remote file system, that is only mounted when needed and unmounted as soon as possible. This ensures sensitive data that are not being actively used are left encrypted in storage. The *D2_SECRETS_BASEDIR* environment variable or *-d* option allow you to specify the directory to use.

The *psec* CLI has a secure deletion mechanism that over-writes file contents prior to deletion, helping to reduce leaving remnants of secrets in unallocated file system storage, similar to the way the Linux *shred* command works.

1.5 Python Script Security

Last, but certainly not least, take the time to read up on Python Security and understand the types and sources of security vulnerabilities related to Python programs. Keep these ideas in mind when using and/or modifying this program.

As part of testing, the Bandit security validation program is used. (See Getting started with Bandit).

In situations where Bandit warnings can safely be ignored, the `# nosec` comment appears on source code lines. Comments as to why these can be safely ignored are included in the code. (Please feel free to issue pull requests if you disagree.)

One runtime security mechanism employed by `psec` is control of the process' `umask`. This is important when running programs that create files, which will inherit their permissions per the process `umask`. The `umask` will be inherited by every new child process and can be set in the user's `.bashrc` (or other shell initialization) file.

The `psec run` command can be used to run programs as child processes, optionally exporting environment variables as well, so controlling the `umask` results in improved file permission security regardless of whether the user knows to set their process `umask`.

You can see the effect in these two examples.

First, by setting the `umask` to `0` you see the very permissive file permissions (as well as getting a warning from `psec` about finding a file with lax permissions):

```
$ psec --umask 0o000 run -- dd if=/dev/random count=1 of=$(psec environments path --
↪tmpdir)/foo
1+0 records in
1+0 records out
512 bytes copied, 0.000019 s, 2.7 MB/s
$ ls -l $(psec environments path --tmpdir)/foo
[!] file /Users/dittrich/.secrets/python_secrets/tmp/foo is mode 0o100666
-rw-rw-rw- 1 dittrich staff 512 Sep  8 13:05 /Users/dittrich/.secrets/python_secrets/
↪tmp/foo
$ rm $(psec environments path --tmpdir)/foo
```

Now when using the default `--umask` value, the file permissions are restricted (and thus no more warning):

```
$ psec run -- dd if=/dev/random count=1 of=$(psec environments path --tmpdir)/foo
1+0 records in
1+0 records out
512 bytes copied, 0.000243 s, 2.1 MB/s
$ ls -l $(psec environments path --tmpdir)/foo
```

(continues on next page)

(continued from previous page)

```
-rw------- 1 dittrich staff 512 Sep  8 13:04 /Users/dittrich/.secrets/python_secrets/
↪tmp/foo
$ rm $(psec environments path --tmpdir)/foo
```

1.6 Bugs, Enhancements, and Future Work

Feature requests (and of course bug reports) are highly encouraged. You can do that by opening an issue on GitHub. Better yet, make a pull request with your own fix or feature. (Check there to see if one may already exist.)

If you want to help, there are some things that are on the "to do" list. These are tracked on this repository's GitHub Projects page.

General or more elaborate potential enhancements are listed here:

- Increase test coverage (test driven development is a Good Thing(TM))
- The Mantl project (GitHub mantl/mantl) employs a security-setup script that takes care of setting secrets (and non-secret related variables) in a monolithic manner. It has specific command line options, specific secret generation functions, and specific data structures for each of the component subsystems used by mantl/mantl. This method is not modular or extensible, and the security-setup script is not generalized such that it can be used by any other project. These limitations are primary motivators for writing `python_secrets`, which could eventually replace `security-setup`.

  At this point, the Mantl `security.yml` file can be read in and values can be manually set, as seen here:

```
$ psec -d ~/git/mantl --secrets-file security.yml secrets show -f yaml
secrets descriptions directory not found
- Value: admin:password
  Variable: chronos_http_credentials
- Value: chronos
  Variable: chronos_principal
- Value: S0JMz5z8oxQGQXMyZjwE0ZCmu4zeJV4oWDUrdc25MBLx
  Variable: chronos_secret
- Value: 88821cbe-c004-4cff-9f91-2bc36cd347dc
  Variable: consul_acl_agent_token
- Value: f9acbe14-28d3-4d06-a1c9-c617da5ebb4e
  Variable: consul_acl_mantl_api_token
- Value: de54ae85-8226-4146-959f-8926b0b8ee55
  Variable: consul_acl_marathon_token
- Value: dfc9b244-5140-41ad-b93a-ac5c2451fb95
  Variable: consul_acl_master_token
- Value: e149b50f-cb5c-4efe-be96-26a52efdc715
  Variable: consul_acl_secure_token
- Value: 719f2328-6446-4647-adf6-310013bac636
  Variable: consul_acl_vault_token
- Value: Z0niD1jeiTkx7xaoewJm2A==
  Variable: consul_gossip_key
- Value: true
  Variable: do_chronos_auth
- Value: true
  Variable: do_chronos_iptables
- Value: true
  Variable: do_chronos_ssl
- Value: true
  Variable: do_consul_auth
```

(continues on next page)

(continued from previous page)

```
- Value: true
  Variable: do_consul_ssl
- Value: true
  Variable: do_mantl_api_auth
- Value: true
  Variable: do_mantlui_auth
- Value: true
  Variable: do_mantlui_ssl
- Value: true
  Variable: do_marathon_auth
- Value: true
  Variable: do_marathon_iptables
- Value: true
  Variable: do_marathon_ssl
- Value: true
  Variable: do_mesos_auth
- Value: true
  Variable: do_mesos_follower_auth
- Value: true
  Variable: do_mesos_framework_auth
- Value: true
  Variable: do_mesos_iptables
- Value: true
  Variable: do_mesos_ssl
- Value: false
  Variable: do_private_docker_registry
- Value: mantl-api
  Variable: mantl_api_principal
- Value: Se4R9nRy8WTAgmU9diJyIPwLYsBU+V1yBxTQumiOriK+
  Variable: mantl_api_secret
- Value: admin:password
  Variable: marathon_http_credentials
- Value: marathon
  Variable: marathon_principal
- Value: +Y5bvIsWliFvcWgbXGWa8kwT6Qf3etogQJe+cK+IV2hX
  Variable: marathon_secret
- Value:
  - principal: marathon
    secret: +Y5bvIsWliFvcWgbXGWa8kwT6Qf3etogQJe+cK+IV2hX
  - principal: chronos
    secret: S0JMz5z8oxQGQXMyZjwE0ZCmu4zeJV4oWDUrdc25MBLx
  - principal: mantl-api
    secret: Se4R9nRy8WTAgmU9diJyIPwLYsBU+V1yBxTQumiOriK+
  Variable: mesos_credentials
- Value: follower
  Variable: mesos_follower_principal
- Value: Q53uAa2mNM0UNe2RUjrX6k7QvK6ojjH1gHXYLcm3Lmfr
  Variable: mesos_follower_secret
- Value: password
  Variable: nginx_admin_password
- Value: true
  Variable: security_enabled
- Value: chronos
  Variable: zk_chronos_user
- Value: JWPO11z4lU5qeilZ
  Variable: zk_chronos_user_secret
- Value: hsr+R6YQBAOXoY84a8ne8bU0opg=
```

(continues on next page)

(continued from previous page)

```
  Variable: zk_chronos_user_secret_digest
- Value: marathon
  Variable: zk_marathon_user
- Value: UBh77ok2svQAqWox
  Variable: zk_marathon_user_secret
- Value: mo2mQGXcsc21zB4wYD18jn+Csks=
  Variable: zk_marathon_user_secret_digest
- Value: mesos
  Variable: zk_mesos_user
- Value: L3t9FEMsXehqeBvl
  Variable: zk_mesos_user_secret
- Value: bHYvGteRBxou4jqJ8XWAYmOmzxs=
  Variable: zk_mesos_user_secret_digest
- Value: super
  Variable: zk_super_user
- Value: 2DyL/n/GLi3Q0pa75z9OjODGZKC1RCaEiKNV1ZXo1Wpk
  Variable: zk_super_user_secret
$ psec -d ~/git/mantl --secrets-file security.yml secrets show -f csv | grep␣
↪nginx_admin_password
secrets descriptions directory not found
"nginx_admin_password","password"
$ psec -d ~/git/mantl --secrets-file security.yml secrets set nginx_admin_
↪password=newpassword
secrets descriptions directory not found
$ psec -d ~/git/mantl --secrets-file security.yml secrets show -f csv | grep␣
↪nginx_admin_password
secrets descriptions directory not found
"nginx_admin_password","newpassword"
```

There are a few things that can be done to use `psec` as a replacement for the `security-setup` script. These include:

- Produce secrets descriptions in a `security.d` directory.
- Remove the variables that are not secrets requiring regeneration for rotation or "break-glass" procedures (e.g., like `chronos_principal`, which is a userID value, and `do_mesos_auth`, which is a boolean flag).
- Break down more complex data structures (specifically, the `mesos_credentials` list of dictionaries with keys `principal` and `secret`). These could instead be discrete variables like `marathon_secret` (which appears to be the secret associated with the invariant "variable" `marathon_principal`).

Note: Alternatively, these kind of variables could be supported by defining a type `invariant` or `string` and prompting the user to provide a new value (using any current value as the default).

1.7 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-pypackage

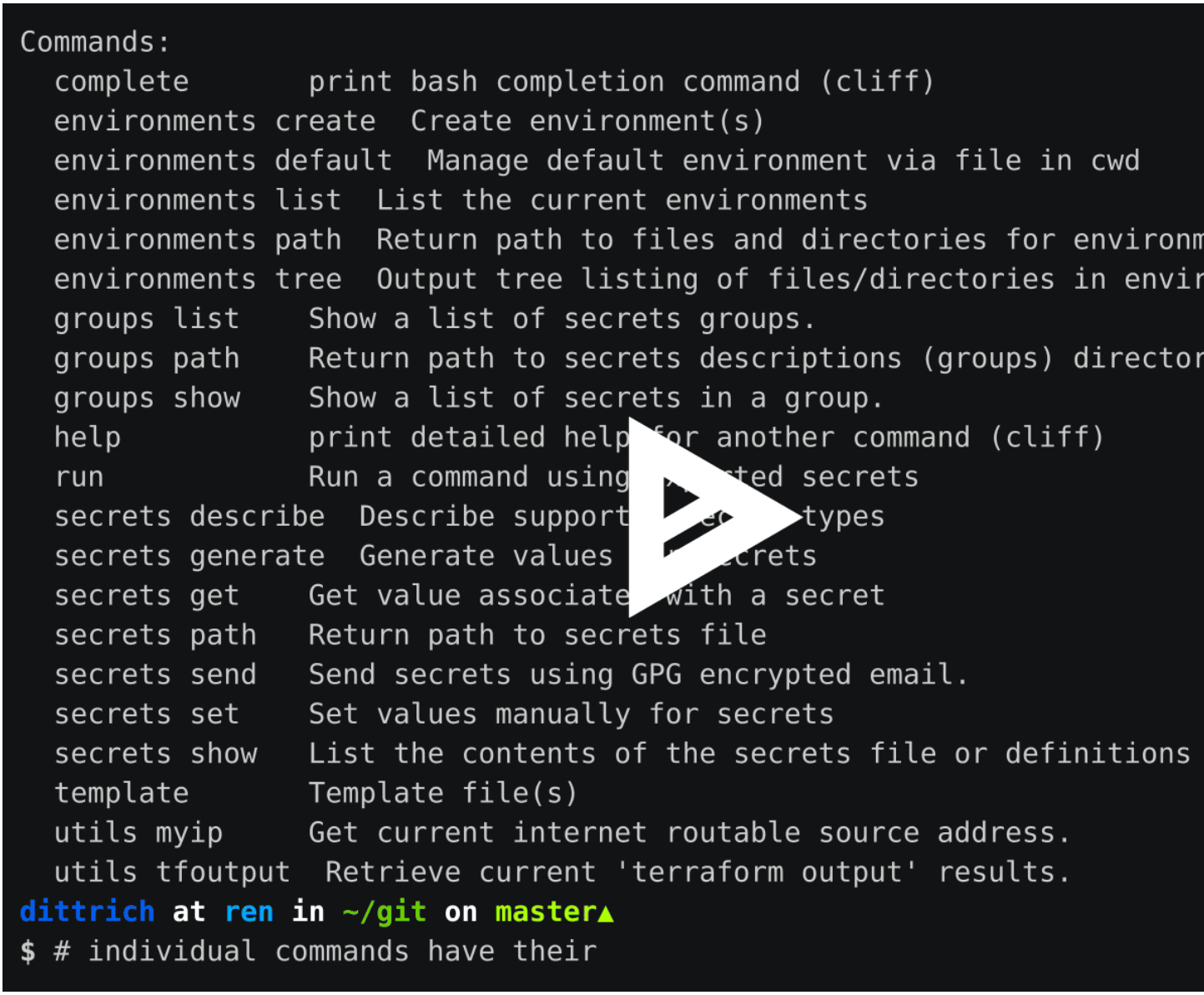
Development of this program was supported in part under an Open Source Development Grant from the Comcast Innovation Fund.

CHAPTER 2

Installation

Install `psec` using the `pip` module of Python 3.6 (or greater):

```
$ python -V
$ Python 3.9.0
$ python -m pip install python_secrets
```



For best results, use a Python `virtualenv` to avoid problems due to the system Python not conforming to the version dependency. User's with Mac OS X systems and Windows systems may want to use `miniconda` to standardize management of your virtual environments across those two operating systems as well as Linux.

If you are not doing a lot of Python development and just want to use `psec` for managing secrets in your open source projects (or as part of an open source project that depends on `psec` for configuration files that include secrets) there are some simpler options that transparently handle the virtual environment creation for you. The `pipx` program is easy to install as a PyPI package, or with Homebrew on the Mac (see https://pipxproject.github.io/pipx/ for installation instructions).

CHAPTER 3

Usage

Commands (and subcommands) generally follow the model set by the OpenStackClient for its Command Structure. The general structure of a command is:

```
$ psec [<global-options>] <object-1> <action> [<object-2>] [<command-arguments>]
```

Note: When originally written, `python_secrets` was the primary command name. And the Python `secrets` module did not yet exist. The name `python_secrets` is a little unwieldy to type on the command line, so a shorter script name `psec` was also included in the original release. The original `python_secrets` program name is now deprecated.

In this documentation and command line help, the program name `psec` is used. (The PyPi package may be renamed at a later date to avoid confusion between the package and the program.)

The actions are things like `list`, `show`, `generate`, `set`, etc.

Note: A proof-of-concept for using the `python_secrets` package in an open source project to eliminate default passwords and keep secrets out of the source code repository directory can be found here:

https://davedittrich.github.io/goSecure/documentation.html

3.1 Getting help

To get help information on global command arguments and options, use the `help` command or `--help` option flag. The usage documentation below will detail help output for each command.

3.2 Formatters

The cliff Command Line Formulation Framework provides a set of formatting options that facilitate accessing and using stored secrets in other applications. Data can be passed directly in a structured format like CSV, or passed directly to programs like Ansible using JSON.

Attention: The formatter options are shown in the `--help` output for individual commands (e.g., `psec secrets show --help`). For the purposes of this chapter, including the lengthy formatter options on every command would be quite repetitive and take up a lot of space. For this reason, the formatter options will be suppressed for commands as documented below. The difference (**WITH** and **WITHOUT** the formatting options) would look like this:

WITH formatting options

3.2.1 secrets show

List the contents of the secrets file or definitions.

The `secrets show` command is used to see variables, their values, and exported environment variables to help in using them in your code, shell scripts, etc. To see more metadata-ish information, such as their group, type, etc., use the `secrets describe` command instead.

To get show a subset of secrets, specify their names as arguments. If you instead want to show all secrets in one or more groups, use the `--group` option and specify the group names as arguments, or to show all secrets of one or more types, use the `--type` option:

```
$ psec secrets show
+------------------------+----------+------------------------+
| Variable               | Value    | Export                 |
+------------------------+----------+------------------------+
| jenkins_admin_password | REDACTED | jenkins_admin_password |
| myapp_app_password     | REDACTED | DEMO_app_password      |
| myapp_client_psk       | REDACTED | DEMO_client_ssid       |
| myapp_client_ssid      | REDACTED | DEMO_client_ssid       |
| myapp_pi_password      | REDACTED | DEMO_pi_password       |
| trident_db_pass        | REDACTED | trident_db_pass        |
| trident_sysadmin_pass  | REDACTED | trident_sysadmin_pass  |
+------------------------+----------+------------------------+
```

Visually finding undefined variables in a very long list can be difficult. You can show just undefined variables with the `--undefined` option.

```
psec secrets show
    [-f {csv,json,table,value,yaml}]
    [-c COLUMN]
    [--quote {all,minimal,none,nonnumeric}]
    [--noindent]
    [--max-width <integer>]
    [--fit-width]
    [--print-empty]
    [--sort-column SORT_COLUMN]
    [--sort-ascending | --sort-descending]
    [-C]
    [-p]
    [--undefined]
    [-g | -t]
    [arg [arg ...]]
```

-f <FORMATTER>, **--format** <FORMATTER>
the output format, defaults to table

-c COLUMN, **--column** COLUMN
specify the column(s) to include, can be repeated to show multiple columns

--quote <QUOTE_MODE>
when to include quotes, defaults to nonnumeric

--noindent
whether to disable indenting the JSON

--max-width <integer>
Maximum display width, <1 to disable. You can also use the CLIFF_MAX_TERM_WIDTH environment variable, but the parameter takes precedence.

--fit-width
Fit the table to the display width. Implied if –max-width greater than 0. Set the environment variable CLIFF_FIT_WIDTH=1 to always enable

--print-empty
Print empty table if there is no data to show.

--sort-column SORT_COLUMN
specify the column(s) to sort the data (columns specified first have a priority, non-existing columns are ignored), can be repeated

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

-C, --no-redact
Do not redact values in output

-p, --prompts
Include prompts

--undefined
Only show variables that are not yet defined

-g, --group
Arguments are groups to show

-t, --type
Arguments are types to show

arg

This command is provided by the python-secrets plugin.

WITHOUT formatting options

3.2.2 secrets show

List the contents of the secrets file or definitions.

The `secrets show` command is used to see variables, their values, and exported environment variables to help in using them in your code, shell scripts, etc. To see more metadata-ish information, such as their group, type, etc., use the `secrets describe` command instead.

To get show a subset of secrets, specify their names as arguments. If you instead want to show all secrets in one or more groups, use the `--group` option and specify the group names as arguments, or to show all secrets of one or more types, use the `--type` option:

```
$ psec secrets show
+-------------------------+----------+-------------------------+
| Variable                | Value    | Export                  |
+-------------------------+----------+-------------------------+
| jenkins_admin_password | REDACTED | jenkins_admin_password |
| myapp_app_password     | REDACTED | DEMO_app_password      |
| myapp_client_psk       | REDACTED | DEMO_client_ssid       |
| myapp_client_ssid      | REDACTED | DEMO_client_ssid       |
| myapp_pi_password      | REDACTED | DEMO_pi_password       |
| trident_db_pass        | REDACTED | trident_db_pass        |
| trident_sysadmin_pass  | REDACTED | trident_sysadmin_pass  |
+-------------------------+----------+-------------------------+
```

Visually finding undefined variables in a very long list can be difficult. You can show just undefined variables with the `--undefined` option.

```
psec secrets show
    [--sort-ascending | --sort-descending]
    [-C]
    [-p]
    [--undefined]
    [-g | -t]
    [arg [arg ...]]
```

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

-C, --no-redact
Do not redact values in output

-p, --prompts
Include prompts

--undefined
Only show variables that are not yet defined

-g, --group
Arguments are groups to show

-t, --type
Arguments are types to show

arg

This command is provided by the python-secrets plugin.

3.2.3 Formatting examples

CSV output (with header) can be produced like this:

```
$ psec secrets show -f csv --column Variable --column Value
"Variable","Value"
"trident_db_pass","gargle earlobe eggplant kissable"
"ca_rootca_password","gargle earlobe eggplant kissable"
"consul_key","HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
"jenkins_admin_password","gargle earlobe eggplant kissable"
"rabbitmq_default_user_pass","gargle earlobe eggplant kissable"
"rabbitmq_admin_user_pass","gargle earlobe eggplant kissable"
"trident_sysadmin_pass","gargle earlobe eggplant kissable"
"vncserver_password","gargle earlobe eggplant kissable"
"zookeeper_uuid4","769a77ad-b06f-4018-857e-23f970c777c2"
```

Or you can produce JSON and have structured data for consumption by other programs.

```
$ psec secrets show -f json --column Variable --column Value
[
  {
    "Variable": "trident_db_pass",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "ca_rootca_password",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "consul_key",
    "Value": "HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
  },
  {
    "Variable": "jenkins_admin_password",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "rabbitmq_default_user_pass",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "rabbitmq_admin_user_pass",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "trident_sysadmin_pass",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "vncserver_password",
    "Value": "gargle earlobe eggplant kissable"
  },
  {
    "Variable": "zookeeper_uuid4",
    "Value": "769a77ad-b06f-4018-857e-23f970c777c2"
  }
]
```

The JSON can be manipulated, filtered, and restructured using a program like `jq`, for example:

```
$ psec secrets show -f json --column Variable --column Value |
```

(continues on next page)

(continued from previous page)

```
> jq -r '.[] | { (.Variable): .Value } '
{
  "trident_db_pass": "gargle earlobe eggplant kissable"
}
{
  "ca_rootca_password": "gargle earlobe eggplant kissable"
}
{
  "consul_key": "HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
}
{
  "jenkins_admin_password": "gargle earlobe eggplant kissable"
}
{
  "rabbitmq_default_user_pass": "gargle earlobe eggplant kissable"
}
{
  "rabbitmq_admin_user_pass": "gargle earlobe eggplant kissable"
}
{
  "trident_sysadmin_pass": "gargle earlobe eggplant kissable"
}
{
  "vncserver_password": "gargle earlobe eggplant kissable"
}
{
  "zookeeper_uuid4": "769a77ad-b06f-4018-857e-23f970c777c2"
}
```

```
$ psec secrets show -f json --column Variable --column Value |
> jq -r '.[] | [ (.Variable), .Value ] '
[
  "trident_db_pass",
  "gargle earlobe eggplant kissable"
]
[
  "ca_rootca_password",
  "gargle earlobe eggplant kissable"
]
[
  "consul_key",
  "HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
]
[
  "jenkins_admin_password",
  "gargle earlobe eggplant kissable"
]
[
  "rabbitmq_default_user_pass",
  "gargle earlobe eggplant kissable"
]
[
  "rabbitmq_admin_user_pass",
  "gargle earlobe eggplant kissable"
]
[
```

(continues on next page)

(continued from previous page)

```
  "trident_sysadmin_pass",
  "gargle earlobe eggplant kissable"
]
[
  "vncserver_password",
  "gargle earlobe eggplant kissable"
]
[
  "zookeeper_uuid4",
  "769a77ad-b06f-4018-857e-23f970c777c2"
]
```

```
$ psec secrets show -f json --column Variable --column Value |
> jq -r '.[] | [ (.Variable), .Value ] |@sh'
'trident_db_pass' 'gargle earlobe eggplant kissable'
'ca_rootca_password' 'gargle earlobe eggplant kissable'
'consul_key' 'HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY='
'jenkins_admin_password' 'gargle earlobe eggplant kissable'
'rabbitmq_default_user_pass' 'gargle earlobe eggplant kissable'
'rabbitmq_admin_user_pass' 'gargle earlobe eggplant kissable'
'trident_sysadmin_pass' 'gargle earlobe eggplant kissable'
'vncserver_password' 'gargle earlobe eggplant kissable'
'zookeeper_uuid4' '769a77ad-b06f-4018-857e-23f970c777c2'
```

```
$ psec secrets show -f json --column Variable --column Value |
> jq -r '.[] | [ (.Variable), .Value ] |@csv'
"trident_db_pass","gargle earlobe eggplant kissable"
"ca_rootca_password","gargle earlobe eggplant kissable"
"consul_key","HEvUAItLFZ0+GjxfwTxLDKq5Fbt86UtXrInzpf71GGY="
"jenkins_admin_password","gargle earlobe eggplant kissable"
"rabbitmq_default_user_pass","gargle earlobe eggplant kissable"
"rabbitmq_admin_user_pass","gargle earlobe eggplant kissable"
"trident_sysadmin_pass","gargle earlobe eggplant kissable"
"vncserver_password","gargle earlobe eggplant kissable"
"zookeeper_uuid4","769a77ad-b06f-4018-857e-23f970c777c2"
```

3.3 Commands

The following is reduced `--help` output for each subcommand supported by `psec`. See *Formatting examples*, or explicitly request `--help` output for the subcommand in question on the command line, to see the suppressed formatting options.

3.3.1 Environments

environments create

Create environment(s).

Empty environments can be created as needed, one at a time or several at once. Specify the names on the command line as arguments:

```
$ psec environments create development testing production
[+] environment directory /Users/dittrich/.secrets/development created
[+] environment directory /Users/dittrich/.secrets/testing created
[+] environment directory /Users/dittrich/.secrets/production created
```

In some special circumstances, it may be necessary to have one set of identical secrets that have different environment names. If this happens, you can create an alias (see also the `environments list` command):

```
$ psec environments create --alias evaluation testing
```

To make it easier to bootstrap an open source project, where the use may not be intimately familiar with all necessary secrets and settings, you can make their life easier by preparing an empty set of secret descriptions that will help prompt the user to set them. You can do this following these steps:

1. Create a template secrets environment directory that contains just the secrets definitions. This example uses the template found in the davedittrich/goSecure repository (directory https://github.com/davedittrich/goSecure/tree/master/secrets).
2. Use this template to clone a secrets environment, which will initially be empty:

   ```
   $ psec environments create test --clone-from ~/git/goSecure/secrets
   [+] new password variable "gosecure_app_password" is unset
   [+] new string variable "gosecure_client_ssid" is unset
   [+] new string variable "gosecure_client_ssid" is unset
   [+] new string variable "gosecure_client_psk" is unset
   [+] new password variable "gosecure_pi_password" is unset
   [+] new string variable "gosecure_pi_pubkey" is unset
   [+] environment directory /Users/dittrich/.secrets/test created
   ```

Note: Directory and file permissions on cloned environments will prevent `other` from having read/write/execute permissions (i.e., `o-rwx` in terms of the `chmod` command.)

```
psec environments create
    [-A ALIAS | -C CLONE_FROM]
    [--force]
    [env [env ...]]
```

-A <ALIAS>, **--alias** <ALIAS>
: Environment to alias

-C <CLONE_FROM>, **--clone-from** <CLONE_FROM>
: Environment directory to clone from

--force
: Create secrets base directory

env

This command is provided by the python-secrets plugin.

environments default

Manage default environment via file in cwd.

If no default is explicitly set, the default that would be applied is returned:

```
$ cd ~/git/psec
$ psec environments default
[+] default environment is "psec"
```

When listing environments, the default environment that would be implicitly used will be identified:

```
$ psec environments list
+-------------+---------+
| Environment | Default |
+-------------+---------+
| development | No      |
| testing     | No      |
| production  | No      |
+-------------+---------+
```

The following shows setting and unsetting the default:

```
$ psec environments default testing
[+] default environment set to "testing"
$ psec environments default
testing
$ psec environments list
+-------------+---------+
| Environment | Default |
+-------------+---------+
| development | No      |
| testing     | Yes     |
| production  | No      |
+-------------+---------+
$ psec environments default --unset-default
[+] default environment unset
```

```
psec environments default [--set | --unset] [environment]
```

--set
: Set localized environment default

--unset
: Unset localized environment default

environment

This command is provided by the python-secrets plugin.

environments delete

Delete environment.

Deleting an environment requires explicitly naming the environment to delete and confirmation from the user. This is done in one of two ways: by prompting the user to confirm the environment to delete, or by requiring the `--force` option flag be set along with the name.

When this command is run in a terminal shell (i.e., with a TTY), the user will be asked to type the name again to confirm the operation:

```
$ psec environments delete testenv
Type the name 'testenv' to confirm: testenv
[+] deleted directory path '/Users/dittrich/.secrets/testenv'
```

If no TTY is present (i.e., a shell script running in the background), an exception is raised that includes the files that will be deleted and explaining how to force the deletion:

```
$ psec environments delete testenv
[-] must use '--force' flag to delete an environment.
[-] the following will be deleted:
/Users/dittrich/.secrets/testenv
├── secrets.d
│   ├── ansible.json
│   ├── ca.json
│   ├── consul.json
│   ├── do.json
│   ├── jenkins.json
│   ├── opendkim.json
│   ├── rabbitmq.json
│   └── trident.json
└── token.json
```

The `--force` flag will allow deletion of the environment:

```
$ psec environments delete --force testenv
[+] deleted directory path /Users/dittrich/.secrets/testenv
```

```
psec environments delete [--force] [environment]
```

`--force`
: Mandatory confirmation

`environment`

This command is provided by the python-secrets plugin.

environments list

List the current environments.

You can get a list of all available environments at any time, including which one would be the default used by sub-commands:

```
$ psec environments list
+-------------+---------+
| Environment | Default |
+-------------+---------+
| development | No      |
| testing     | No      |
| production  | No      |
+-------------+---------+
```

To see which environments are aliases, use the `--aliasing` option:

```
$ psec -v environments create --alias evaluation testing
$ psec environments list --aliasing
+-------------+---------+-----------+
| Environment | Default | AliasFor |
+-------------+---------+-----------+
| development | No      |           |
| evaluation  | No      | testing  |
```

(continues on next page)

(continued from previous page)

```
| testing      | No      |          |
| production   | No      |          |
+--------------+---------+----------+
```

If there are any older environments that contain `.yml` files for storing secrets or definitions, they will be called out when you list environments. (Adding `-v` will explicitly list the names of files that are found if you wish to see them.):

```
$ psec environments list
[!] environment 'algo' needs conversion (see 'psec utils yaml-to-json --help')
[!] environment 'hypriot' needs conversion (see 'psec utils yaml-to-json --help')
[!] environment 'kali-packer' needs conversion (see 'psec utils yaml-to-json --help')
+-------------------------+---------+
| Environment             | Default |
+-------------------------+---------+
| attack_range            | No      |
| attack_range_local      | No      |
| flash                   | No      |
| python_secrets          | Yes     |
+-------------------------+---------+
```

```
psec environments list
    [--sort-ascending | --sort-descending]
    [--aliasing]
```

--sort-ascending
: sort the column(s) in ascending order

--sort-descending
: sort the column(s) in descending order

--aliasing
: Include aliasing

This command is provided by the python-secrets plugin.

environments path

Return path to files and directories for environment.

Provides the full absolute path to the environment directory for the environment and any specified subdirectories:

```
$ psec environments path
/Users/dittrich/.secrets/psec
$ psec environments path -e goSecure
/Users/dittrich/.secrets/goSecure
```

Using the `--exists` option will just exit with return code `0` when the environment directory exists, or `1` if it does not, and no path is printed on stdout.

To append subdirectory components, provide them as arguments and they will be concatenated with the appropriate OS path separator:

```
$ psec environments path -e goSecure configs
/Users/dittrich/.secrets/goSecure/configs
```

To ensure the directory path specified by command line arguments is present in the file system, use the `--create` option.

Using the `--tmpdir` option will return the path to the temporary directory for the environment. If the environment's directory already exists, the temporary directory will be also be created so it is ready for use. If the environment directory does not already exist, the program will exit with an error message. Again, the `--create` changes this behavior and the missing directory path will be created.

```
psec environments path
    [--create]
    [--exists]
    [--json]
    [--tmpdir]
    [subdir [subdir ...]]
```

`--create`
: Create the directory path if it does not yet exist

`--exists`
: Check to see if environment exists and return exit code (0==exists, 1==not)

`--json`
: Output in JSON (e.g., for Terraform external data source)

`--tmpdir`
: Create and/or return tmpdir for this environment

`subdir`

This command is provided by the python-secrets plugin.

environments rename

Rename environment.

Just like *mv*, renames an environment from the name specified by the first argument to that of the second argument:

```
$ psec environments list
+----------------+---------+
| Environment    | Default |
+----------------+---------+
| old            | No      |
+----------------+---------+
$ psec environments rename new old
[-] source environment "new" does not exist
$ psec environments rename old new
[+] environment "old" renamed to "new"
$ psec environments list
+----------------+---------+
| Environment    | Default |
+----------------+---------+
| new            | No      |
+----------------+---------+
```

```
psec environments rename source dest
```

`source`
: environment to rename

`dest`
: new environment name

This command is provided by the python-secrets plugin.

environments tree

Output tree listing of files/directories in environment.

The `environments tree` command produces output similar to the Unix `tree` command:

```
$ psec -e d2 environments tree
/Users/dittrich/.secrets/d2
├── backups
│   ├── black.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-06T23:36:58PDT.tgz
│   │   └── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   ├── green.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-06T23:45:49PDT.tgz
│   │   └── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   ├── purple.secretsmgmt.tk
│   │   ├── letsencrypt_2018-04-25T16:32:20PDT.tgz
│   │   ├── trident_2018-01-31T23:38:48PST.tar.bz2
│   │   └── trident_2018-02-04T20:05:33PST.tar.bz2
│   └── red.secretsmgmt.tk
│       ├── letsencrypt_2018-04-06T23:45:49PDT.tgz
│       └── letsencrypt_2018-04-25T16:32:20PDT.tgz
├── dittrich.asc
├── keys
│   └── opendkim
│       └── secretsmgmt.tk
│           ├── 201801.private
│           ├── 201801.txt
│           ├── 201802.private
│           └── 201802.txt
├── secrets.d
│   ├── ca.json
│   ├── consul.json
│   ├── jenkins.json
│   ├── rabbitmq.json
│   ├── trident.json
│   ├── vncserver.json
│   └── zookeper.json
├── secrets.json
└── vault_password.txt
```

To just see the directory structure and not files, add the `--no-files` option:

```
$ psec -e d2 environments tree --no-files
/Users/dittrich/.secrets/d2
├── backups
│   ├── black.secretsmgmt.tk
│   ├── green.secretsmgmt.tk
│   ├── purple.secretsmgmt.tk
│   └── red.secretsmgmt.tk
├── keys
│   └── opendkim
│       └── secretsmgmt.tk
└── secrets.d
```

```
psec environments tree [--no-files] [environment]
```

--no-files
Do not include files in listing

environment

This command is provided by the python-secrets plugin.

3.3.2 Groups

groups create

Create a secrets descriptions group.

Secrets and variables are described in files in a drop-in style directory ending in `.d`. This forms 'groups' that organize secrets and variables by purpose, by open source tool, etc. This command creates a new group descriptions file in the selected environment.

When integrating a new open source tool or project with an existing tool or project, you can create a new group in the current environment and clone its secrets descriptions from pre-existing definitions. This does not copy any values, just the descriptions, allowing you to manage the values independently of other projects using a different environment:

```
$ psec groups create newgroup --clone-from ~/git/goSecure/secrets.d/gosecure.json
[+] created new group 'newgroup'
$ psec groups list
+----------+-------+
| Group    | Items |
+----------+-------+
| jenkins  |     1 |
| myapp    |     4 |
| newgroup |    12 |
| trident  |     2 |
+----------+-------+
```

Note: Directory and file permissions on cloned groups will prevent `other` from having read/write/execute permissions (i.e., `o-rwx` in terms of the `chmod` command.)

```
psec groups create [-C CLONE_FROM] [arg]
```

-C <CLONE_FROM>, **--clone-from** <CLONE_FROM>
Group descriptions file to clone from

arg

This command is provided by the python-secrets plugin.

groups delete

Delete a secrets descriptions group.

Deletes a group of secrets and variables by removing them from the secrets environment and deleting their descriptions file.

If the `--force` option is not specified, you will be prompted to confirm the group name before it is deleted.

```
psec groups delete [--force] [group]
```

--force
Mandatory confirmation

group

This command is provided by the python-secrets plugin.

groups list

Show a list of secrets groups.

The names of the groups and number of items are printed by default:

```
$ psec groups list
+---------+-------+
| Group   | Items |
+---------+-------+
| jenkins |     1 |
| myapp   |     4 |
| trident |     2 |
+---------+-------+
```

```
psec groups list [--sort-ascending | --sort-descending]
```

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

This command is provided by the python-secrets plugin.

groups path

Return path to secrets descriptions (groups) directory.

```
$ psec groups path
/Users/dittrich/.secrets/psec/secrets.d
```

```
psec groups path [environment]
```

environment

This command is provided by the python-secrets plugin.

groups show

Show a list of secrets in a group.

Show the group name and number of items in the group for one or more groups:

```
$ psec groups show trident myapp
+---------+-----------------------+
| Group   | Variable              |
+---------+-----------------------+
| trident | trident_sysadmin_pass |
| trident | trident_db_pass       |
| myapp   | myapp_pi_password     |
| myapp   | myapp_app_password    |
| myapp   | myapp_client_psk      |
| myapp   | myapp_client_ssid     |
+---------+-----------------------+
```

```
psec groups show
    [--sort-ascending | --sort-descending]
    [group [group ...]]
```

--sort-ascending
: sort the column(s) in ascending order

--sort-descending
: sort the column(s) in descending order

group

This command is provided by the python-secrets plugin.

3.3.3 Init

init

Initialize a psec secrets base directory.

The *psec* program stores secrets and variables for environments in their own subdirectory trees beneath a top level directory root referred to as the "secrets base directory" (*secrets_basedir*). This directory tree should not be a "normal" file system directory that includes arbitrary files and directories, but rather a special location dedicated to *only* storing secrets environments and related files.

For added security, you can root this directory tree within an encrypted USB-connected disk device, SD card, or other external or remote file system that is only mounted when needed. This ensures sensitive data that are not being actively used are left encrypted in storage. The *D2_SECRETS_BASEDIR* environment variable or *-d* option allow you to specify the directory to use.

To attempt to prevent accidentally storing secrets in directories that are already storing normal files or directories, a special marker file must be present. The *init* command ensures that this secrets base directory is created and marked by the presence of that special file. Until this is done, some *psec* commands may report the base directory is not found (if it does not exist) or is not valid (if it does exist, but does not contain the special marker file):

```
$ psec -d /tmp/foo/does/not/exist environments list
[-] directory '/tmp/foo/does/not/exist' does not exist
```

```
psec init [basedir]
```

basedir

This command is provided by the python-secrets plugin.

3.3.4 Run

run

Run a command line as a sub-shell.

The `run` subcommand is used to run a command line in a sub-shell similar to using `bash -c`.

When used with the `--elapsed` option, you get more readable elapsed time information than with the `time` command:

```
$ psec --elapsed run sleep 3
[+] elapsed time 00:00:03.01
```

When combined with the `-E` option to export a `psec` environment's secrets and variables into the process environment, the command sub-shell (and every shell that is subsequently forked & exec'd) will inherit these environment variables. Programs like Ansible and Terraform can then access the values by reference rather than requiring hard-coding values or passing values on the command line. You can even run a shell program like `bash` or `byobu` in a nested shell to change default values for interactive sessions.

Secrets and variables may be exported with their name, or may have an additional environment variable name (for programs that expect a particular prefix, like `TF_` for Terraform variables) as seen here:

```
$ psec secrets show myapp_pi_password --no-redact
+------------------+--------------------------+------------------+
| Variable         | Value                    | Export           |
+------------------+--------------------------+------------------+
| myapp_pi_password | GAINS.ranged.ENGULF.wound | DEMO_pi_password |
+------------------+--------------------------+------------------+
```

Without the `-E` option the export variable is not set:

```
$ psec run -- bash -c 'env | grep DEMO_pi_password'
$ psec run -- bash -c 'echo The demo password is ${DEMO_pi_password:-not set}'
The demo password is not set
```

With the `-E` option it is set and the sub-shell can expand it:

```
$ psec -E run -- bash -c 'env | grep DEMO_pi_password'
DEMO_pi_password=GAINS.ranged.ENGULF.wound
$ psec run -- bash -c 'echo The demo password is ${DEMO_pi_password:-not set}'
The demo password is GAINS.ranged.ENGULF.wound
```

NOTE: The `--` you see in these examples is necessary to ensure that command line parsing by the shell to construct the argument vector for passing to the `psec` program is stopped so that the options meant for the sub-command are passed to it properly for parsing. Failing to add the `--` may result in a strange parsing error message, or unexpected behavior when the command line you typed is not parsed the way you assumed it would be:

```
$ psec run bash -c 'env | grep DEMO_pi_password'
usage: psec run [-h] [arg [arg ...]]
psec run: error: unrecognized arguments: -c env | grep DEMO_pi_password
```

You may use `--elapsed` without an environment if you do not need to export variables, but when the `-e` option is present an environment must exist or you will get an error.

If no arguments are specified, this `--help` text is output.

```
psec run [arg [arg ...]]
```

arg
Command arguments

This command is provided by the python-secrets plugin.

3.3.5 Secrets

secrets backup

Back up just secrets and descriptions.

Creates a backup (`tar` format) of the secrets.json file and all description files.

```
psec secrets backup
```

This command is provided by the python-secrets plugin.

secrets create

Create a new secret definition.

Defines one or more secrets in a specified group based on input from the user. Secret definitions are created in the user's environments storage directory and a new variable with no value is created there, too.

If the environment and/or the group does not exist, you will be prompted to create them. Use the `--force` option to create them without asking.

To maintain a copy of the secrets descriptions in the source repository so they can be used to quickly configure a new deployment after cloning, use the `--mirror-locally` option when creating secrets from the root of the repository directory. A copy of each modified group description file will be mirrored into a subdirectory tree in the current working directory where you can commit it to the repository.

If no group is specified with the `--group` option, the environment identifier will be used as a default. This simplifies things for small projects that don't need the drop-in style group partitioning that is more appropriate for multi-tool open source system integration where a single monolithic configuration file becomes unwieldy and inflexible. This feature can also be used for "global" variables that could apply across sub-components:

```
$ psec secrets create newsecret --force
```

KNOWN LIMITATION: This subcommand currently only works interactively and you will be prompted for all attributes. In future, this may be handled using `key=value` pairs for attributes, similar to the way the `secrets set` command works.

```
psec secrets create
    [--group GROUP]
    [--force]
    [--update]
    [--mirror-locally]
    [arg [arg ...]]
```

--group <GROUP>
Group in which to define the secret(s)

--force
Create missing environment and/or group

--update
Update the fields in an existing description

--mirror-locally
Mirror definitions locally

arg

This command is provided by the python-secrets plugin.

secrets delete

Delete secrets and their definitions.

Deletes one or more secrets and their definitions from an environment. Unless the `--force` flag is specified, you will be prompted to type in the variable name again to ensure you really want to remove all trace of it from the environment:

```
$ psec secrets delete --group myapp myapp_client_psk myapp_client_ssid
Type the name 'myapp_client_psk' to confirm: myapp_client_psk
Type the name 'myapp_client_ssid' to confirm: myapp_client_ssid
```

If you delete all of the variable descriptions remaining in a group, the group file will be deleted.

The `--mirror-locally` option will manage a local copy of the descriptions file. Use this if you are eliminating a variable from a project while editing files in the root of the source repository.

KNOWN LIMITATION: You must specify the group with the `--group` option currently and are restricted to deleting variables from one group at a time.

```
psec secrets delete
    [-g GROUP]
    [--force]
    [--mirror-locally]
    [arg [arg ...]]
```

-g <GROUP>, **--group** <GROUP>
Group from which to delete the secret(s)

--force
Mandatory confirmation

--mirror-locally
Mirror definitions locally

arg

This command is provided by the python-secrets plugin.

secrets describe

Describe supported secret types.

To get descriptions for a subset of secrets, specify their names as the arguments:

```
$ psec secrets describe jenkins_admin_password
+-----------------------+---------+----------+-------------------------------------
↪+----------+
| Variable              | Group   | Type     | Prompt                                ␣
↪| Options |
```

(continues on next page)

(continued from previous page)

```
+------------------------+---------+----------+-------------------------------------
↪+---------+
| jenkins_admin_password | jenkins | password | Password for Jenkins 'admin' account␣
↪| *       |
+------------------------+---------+----------+-------------------------------------
↪+---------+
```

If you instead want to get descriptions of all secrets in one or more groups, use the `--group` option and specify the group names as the arguments.

To instead see the values and exported environment variables associated with secrets, use the `secrets show` command instead.

```
psec secrets describe
    [--sort-ascending | --sort-descending]
    [--undefined]
    [-g | -t]
    [arg [arg ...]]
```

--sort-ascending
: sort the column(s) in ascending order

--sort-descending
: sort the column(s) in descending order

--undefined
: Only show variables that are not yet defined

-g, --group
: Arguments are groups to list

-t, --types
: Describe types

arg

This command is provided by the python-secrets plugin.

secrets find

Find defined secrets in environments.

Searches through all environments in the secrets base directory and lists those that contain the variable(s) with names matching the search terms. You can search for secrets by value instead using the *–value* option flag. Example:

```
$ psec secrets find tanzanite_admin
[+] searching secrets base directory /Users/dittrich/.secrets
+-------------+-----------+----------------------------+
| Environment | Group     | Variable                   |
+-------------+-----------+----------------------------+
| tanzanite   | tanzanite | tanzanite_admin_user_email |
| tanzanite   | tanzanite | tanzanite_admin_password   |
| tzdocker    | tanzanite | tanzanite_admin_user_email |
| tzdocker    | tanzanite | tanzanite_admin_password   |
| tztest      | tanzanite | tanzanite_admin_user_email |
| tztest      | tanzanite | tanzanite_admin_password   |
+-------------+-----------+----------------------------+
```

```
psec secrets find
    [--sort-ascending | --sort-descending]
    [--group GROUP]
    [--value]
    [arg [arg ...]]
```

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

--group <GROUP>
Limit searches to this specific group

--value
The search term is the value to find (not the variable name)

arg

This command is provided by the python-secrets plugin.

secrets generate

Generate values for secrets.

Sets variables by generating values according to the `Type` definition for each variable.

If you include the `--from-options` flag, string variables will also be set according to their default value as described in the help output for the `secrets set` command. This allows as many variables as possible to be set with a single command (rather than requiring the user to do both `secrets set` and `secrets generate` as two separate steps.

To affect only a subset of secrets, specify their names as the arguments to this command. If no secrets are specified, all secrets will be affected.

```
psec secrets generate
    [--from-options]
    [-U]
    [--min-words-length MIN_WORDS_LENGTH]
    [--max-words-length MAX_WORDS_LENGTH]
    [--min-acrostic-length MIN_ACROSTIC_LENGTH]
    [--max-acrostic-length MAX_ACROSTIC_LENGTH]
    [--acrostic ACROSTIC]
    [--delimiter DELIMITER]
    [-C CASE]
    [arg [arg ...]]
```

--from-options
Set variables from first available option

-U, --unique
Generate unique values for each type of secret

--min-words-length <MIN_WORDS_LENGTH>
Minimum word length for XKCD words list

--max-words-length <MAX_WORDS_LENGTH>
Maximum word length for XKCD words list

--min-acrostic-length <MIN_ACROSTIC_LENGTH>
Minimum length of acrostic word for XKCD password

--max-acrostic-length <MAX_ACROSTIC_LENGTH>
Maximum length of acrostic word for XKCD password

--acrostic <ACROSTIC>
Acrostic word for XKCD password

--delimiter <DELIMITER>
Delimiter for XKCD password

-C CASE, **--case** CASE
Choose the method for setting the case of each word in the passphrase. Choices: [‘alternating’, ‘upper’, ‘lower’, ‘random’, ‘first’, ‘capitalize’]

arg

This command is provided by the python-secrets plugin.

secrets get

Gets the value associated with a secret and returns it on standard output. This is used for inline command substitution in cases where a single value is needed:

```
$ echo "Jenkins admin password: $(psec secrets get jenkins_admin_password)"
Jenkins admin password: OZONE.negate.TIPTOP.ocean
```

To get values for more than one secret, use *secrets show* with one of the formatting options allowing you to parse the results or otherwise use them as a group.

```
psec secrets get [-C] secret
```

-C, --content
Get content if secret is a file path

secret

This command is provided by the python-secrets plugin.

secrets path

Return path to secrets file.

If no arguments are present, the path to the secrets for the default environment is returned. If you want to get the secrets path for a specific environment, specify it as the argument to this command.

```
psec secrets path [environment]
```

environment

This command is provided by the python-secrets plugin.

secrets restore

Restore secrets and descriptions from a backup file.

```
psec secrets restore [backup]
```

backup

This command is provided by the python-secrets plugin.

secrets send

Send secrets using GPG encrypted email.

Recipients for the secrets are specified as `USERNAME@EMAIL.ADDRESS` strings and/or `VARIABLE` references.

```
psec secrets send
    [-T]
    [--test-smtp]
    [-H SMTP_HOST]
    [-U SMTP_USERNAME]
    [-F SMTP_SENDER]
    [-S SMTP_SUBJECT]
    [arg [arg ...]]
```

-T, --refresh-token
Refresh Google API Oauth2 token and exit

--test-smtp
Test Oauth2 SMTP authentication and exit

-H <SMTP_HOST>, **--smtp-host** <SMTP_HOST>
SMTP host

-U <SMTP_USERNAME>, **--smtp-username** <SMTP_USERNAME>
SMTP authentication username

-F <SMTP_SENDER>, **--from** <SMTP_SENDER>
Sender address

-S <SMTP_SUBJECT>, **--subject** <SMTP_SUBJECT>
Subject line

arg

This command is provided by the python-secrets plugin.

secrets set

Set values manually for secrets.

One or more secrets can be set directly by specifying them as `variable=value` pairs as the arguments to this command:

```
$ psec secrets set trident_db_pass="rural coffee purple sedan"
```

Adding the `--undefined` flag will limit the secrets being set to only those that are currently not set. If values are not set, you are prompted for the value.

When cloning an environment from definitions in a source repository or an existing environment, you can set secrets by copying them from another existing environment using the `--from-environment` option:

```
$ psec secrets set gosecure_pi_password --from-environment goSecure
```

When you are doing this immediately after cloning (when all variables are undefined) you can set all undefined variables at once from another environment this way:

```
$ psec environments create --clone-from goSecure
$ psec secrets set --undefined --from-environment goSecure
```

To facilitate setting variables from another environment where the variable names may differ, use the assignment style syntax for arguments along with the `--from-environment` option, like this:

```
$ psec secrets set hypriot_client_psk=gosecure_client_psk \
> hypriot_client_ssid=gosecure_client_ssid \
> --from-environment goSecure
```

If you do not provide values for variables using assignment syntax and you are not copying values from another environment, you will be prompted for values according to how the options field is defined.

- If the *only* option is `*` (meaning "any string"), you will be prompted to enter a value. When prompted this way, you can cancel setting the variable by entering an empty string. If you really want the value to be an empty string, you *must* use the assignment syntax with an empty string like this: `variable=''`
- An options list that *does not contain* a `*` defines a finite set of options. This means you are resticted to *only* choosing from the list. This is similar to the `Boolean` type, which can only have a value of `true` or `false`.
- If one or more options are listed *along with* `*`, you can either choose from one of the listed values or select `*` to manually enter a value not in the list.
- You can back out of making a change by selecting `<CANCEL>` from the list.

```
psec secrets set
    [--undefined]
    [--ignore-missing]
    [--from-environment <environment> | --from-options]
    [arg [arg ...]]
```

--undefined
: Set values for undefined variables

--ignore-missing
: Skip setting variables that are not defined

--from-environment `<environment>`
: Environment from which to copy secret value(s)

--from-options
: Set from first available option

arg

This command is provided by the python-secrets plugin.

secrets show

List the contents of the secrets file or definitions.

The `secrets show` command is used to see variables, their values, and exported environment variables to help in using them in your code, shell scripts, etc. To see more metadata-ish information, such as their group, type, etc., use the `secrets describe` command instead.

To get show a subset of secrets, specify their names as arguments. If you instead want to show all secrets in one or more groups, use the `--group` option and specify the group names as arguments, or to show all secrets of one or more types, use the `--type` option:

```
$ psec secrets show
+------------------------+----------+------------------------+
| Variable               | Value    | Export                 |
+------------------------+----------+------------------------+
| jenkins_admin_password | REDACTED | jenkins_admin_password |
| myapp_app_password     | REDACTED | DEMO_app_password      |
| myapp_client_psk       | REDACTED | DEMO_client_ssid       |
| myapp_client_ssid      | REDACTED | DEMO_client_ssid       |
| myapp_pi_password      | REDACTED | DEMO_pi_password       |
| trident_db_pass        | REDACTED | trident_db_pass        |
| trident_sysadmin_pass  | REDACTED | trident_sysadmin_pass  |
+------------------------+----------+------------------------+
```

Visually finding undefined variables in a very long list can be difficult. You can show just undefined variables with the `--undefined` option.

```
psec secrets show
    [--sort-ascending | --sort-descending]
    [-C]
    [-p]
    [--undefined]
    [-g | -t]
    [arg [arg ...]]
```

--sort-ascending
: sort the column(s) in ascending order

--sort-descending
: sort the column(s) in descending order

-C, --no-redact
: Do not redact values in output

-p, --prompts
: Include prompts

--undefined
: Only show variables that are not yet defined

-g, --group
: Arguments are groups to show

-t, --type
: Arguments are types to show

arg

This command is provided by the python-secrets plugin.

secrets tree

Output tree listing of groups and secrets in environment.

The `secrets tree` command produces output similar to the Unix `tree` command. This gives you a visual overview of the groupings of secrets in the target environment:

```
$ psec secrets tree my_environment
my_environment
├── myapp
│   ├── myapp_app_password
│   ├── myapp_client_psk
│   ├── myapp_client_ssid
│   ├── myapp_ondemand_wifi
│   └── myapp_pi_password
└── oauth
    ├── google_oauth_client_id
    ├── google_oauth_client_secret
    ├── google_oauth_refresh_token
    └── google_oauth_username
```

```
psec secrets tree [environment]
```

environment

This command is provided by the python-secrets plugin.

3.3.6 SSH

ssh config

Create SSH configuration snippets for use by `update-dotdee`.

Names of the snippets will start with `psec_` followed by the environment name and host name (separated by _) in order to support global removal of existing snippets for a given environment, while avoiding name clashes with other environments.

These configuration snippets will reside in the directory `~/.ssh/config.d/` and `update-dotdee` will be run after creating them to apply the new configuration(s) for immediate use.

To ensure that no inactive instances still have configurations, the `--clean` option will delete all existing snippets with the prefix for this environment prior to creating new files.

Use `-vvv` to see the configuration file in the terminal command line output.

```
psec ssh config
    [--clean]
    [--public-ip PUBLIC_IP]
    [--known-hosts-root KNOWN_HOSTS_ROOT]
    [--ssh-user SSH_USER]
    [--public-dns PUBLIC_DNS]
    [--show-config]
```

--clean
: Clean out all existing snippets before writing new files

--public-ip <PUBLIC_IP>
: IP address of host

--known-hosts-root <KNOWN_HOSTS_ROOT>
: Root for extracted known_hosts files

--ssh-user <SSH_USER>
: SSH user account

--public-dns <PUBLIC_DNS>
Public DNS name of host

--show-config
Show the SSH configuration on standard output and exit

This command is provided by the python-secrets plugin.

ssh known-hosts add

Add public SSH keys to known_hosts file(s).

This command will either extract SSH public host keys and fingerprints from a cloud service console output (either directly via an API or or from saved output from instantiation of the cloud instance.)

By default, the public keys are added to the system `known_hosts` file (`/etc/ssh/ssh_known_hosts`, which is not writeable by normal users) for added security. The file is manipulated indirectly using an embedded Ansible playbook.

Use `--show-playbook` to just see the Ansible playbook without running it. Use `-vvv` to see the Ansible playbook while it is being applied.

The Ansible playbook uses `become` to elevate privileges. On Linux systems, this usually relies on `sudo`. If plays in the playbook fail with `"module_stderr": "sudo: a password is required"` in the message, you will need to use the `--ask-become-pass` option to be prompted for your password.

NOTE: Because of the use of Ansible, with the potential need for using `sudo`, you cannot pipe console-output text into `psec` using I/O redirection or piping. The following exception will be thrown:

```
...
File "[...]/site-packages/pexpect/pty_spawn.py", line 783, in interact
  mode = tty.tcgetattr(self.STDIN_FILENO)
termios.error: (25, 'Inappropriate ioctl for device')
...
```

If this happens, save the console output to a file and pass its name as a command line argument.

```
psec ssh known-hosts add
    [--known-hosts-root KNOWN_HOSTS_ROOT]
    [--public-ip PUBLIC_IP]
    [--public-dns PUBLIC_DNS]
    [--instance-id INSTANCE_ID]
    [--ask-become-pass]
    [--show-playbook]
    [--save-to-files]
    [source]
```

--known-hosts-root <KNOWN_HOSTS_ROOT>
Root for extracted known_hosts files

--public-ip <PUBLIC_IP>
IP address of host

--public-dns <PUBLIC_DNS>
Public DNS name of host

--instance-id <INSTANCE_ID>
instance ID for getting direct AWS console output

--ask-become-pass
Ask for sudo password for Ansible privilege escalation

--show-playbook
Show the playbook on standard output and exit

--save-to-files
Write extracted fingerprints and public keys out to files

source
console output to process

This command is provided by the python-secrets plugin.

ssh known-hosts extract

Extract SSH keys from cloud provider console logs.

Log output may come from stdout (as it does with Terraform), or it may need to be extracted (e.g., after Pulumi creates AWS instances, you need to manually extract the instance log(s) in order to post-process them.

```
psec ssh known-hosts extract
    [--known-hosts-root KNOWN_HOSTS_ROOT]
    [source]
```

--known-hosts-root <KNOWN_HOSTS_ROOT>
Root for extracted known_hosts files

source
console output to process

This command is provided by the python-secrets plugin.

ssh known-hosts remove

Remove SSH keys from known_hosts file(s).

This command indirectly manipulates the known hosts file using an embedded Ansible playbook.

Use `--show-playbook` to just see the Ansible playbook without running it. Use `-vvv` to see the Ansible playbook while it is being applied.

The Ansible playbook uses `become` to elevate privileges. On Linux systems, this usually relies on `sudo`. If plays in the playbook fail with `"module_stderr": "sudo: a password is required"` in the message, you will need to use the `--ask-become-pass` option to be prompted for your password.

NOTE: Because of the use of Ansible, with the potential need for using `sudo`, you cannot pipe console-output text into `psec` using I/O redirection or piping. The following exception will be thrown:

```
...
File "[...]/site-packages/pexpect/pty_spawn.py", line 783, in interact
  mode = tty.tcgetattr(self.STDIN_FILENO)
termios.error: (25, 'Inappropriate ioctl for device')
...
```

If this happens, save the console output to a file and pass its name as a command line argument.

```
psec ssh known-hosts remove
    [--known-hosts-root KNOWN_HOSTS_ROOT]
    [--ask-become-pass]
    [--show-playbook]
    [source]
```

--known-hosts-root <KNOWN_HOSTS_ROOT>
Root for extracted known_hosts files

--ask-become-pass
Ask for sudo password for Ansible privilege escalation

--show-playbook
Show the playbook on standard output and exit

source
console output to process

This command is provided by the python-secrets plugin.

3.3.7 Template

template

Template file(s).

For information on the Jinja2 template engine and how to use it, see http://jinja.pocoo.org

To assist debugging, use `--check-defined` to check that all required variables are defined.

```
psec template [--check-defined] [--no-env] [source] [dest]
```

--check-defined
Just check for undefined variables

--no-env
Do not require and load an environment

source
input Jinja2 template source

dest
templated output destination ('-' for stdout)

This command is provided by the python-secrets plugin.

3.3.8 Utils

utils myip

Get currently active internet routable IPv4 address.

Return the routable IP address of the host running this script using one of several publicly available free methods typically using HTTPS or DNS.

The `--cidr` option expresses the IP address as a CIDR block to use in setting up firewall rules for this specific IP address.

The `--netblock` option follows this lookup with another lookup using WHOIS to get the network provider's address range(s), in CIDR notation, to help with creating firewall rules that can work around dynamic addressing. This is not the most secure way to grant network access as it allows any customer using the same provider to also communicate through the firewall, but you have to admit that it is better than `allow ANY`! ¯_()_/¯

To see a table of the methods, use `utils myip methods`.

KNOWN LIMITATION: Some of the methods may not fully support IPv6 at this point. If you find one that doesn't work, try a different one.

See also: https://linuxize.com/post/how-to-find-ip-address-linux/ https://dev.to/adityathebe/a-handy-way-to-know-your-public-ip-address-with-dns-servers-4nmn

```
psec utils myip
    [-M {akamai,amazon,google,opendns_h,opendns_r,icanhazip,infoip,tnx,random}]
    [-C | -N]
```

-M <METHOD>, **--method** <METHOD>
Method to use for determining IP address

-C, --cidr
Express the IP address as a CIDR block

-N, --netblock
Return network CIDR block(s) for IP from WHOIS

WARNING: Any of the sites used by this command may limit the number of queries allowed from a given source in a given period of time and may temporarily block or reject attempts to use their service beyond the quota limit.

utils myip methods

Show methods for obtaining routable IP address.

Provides the details of the methods coded into this app for obtaining this host's routable IP address:

```
$ psec utils myip methods
+-----------+-------+-----------------------------------------------------+
| Method    | Type  | Source                                              |
+-----------+-------+-----------------------------------------------------+
| akamai    | dns   | dig +short @ns1-1.akamaitech.net ANY whoami.akamai.net |
| amazon    | https | https://checkip.amazonaws.com                       |
| google    | dns   | dig +short @ns1.google.com TXT o-o.myaddr.l.google.com |
| icanhazip | https | https://icanhazip.com/                              |
| infoip    | https | https://api.infoip.io/ip                            |
| opendns_h | https | https://diagnostic.opendns.com/myip                 |
| opendns_r | dns   | dig +short @resolver1.opendns.com myip.opendns.com -4  |
| tnx       | https | https://tnx.nl/ip                                   |
+-----------+-------+-----------------------------------------------------+
```

It can be used for looping in tests, etc. like this:

```
$ for method in $(psec utils myip methods -f value -c Method)
> do
>   echo "$method: $(psec utils myip --method $method)"
> done
akamai: 93.184.216.34
amazon: 93.184.216.34
google: 93.184.216.34
```

(continues on next page)

(continued from previous page)

```
icanhazip: 93.184.216.34
infoip: 93.184.216.34
opendns_h: 93.184.216.34
opendns_r: 93.184.216.34
tnx: 93.184.216.34
```

```
psec utils myip methods
    [--sort-ascending | --sort-descending]
    [method [method ...]]
```

--sort-ascending
: sort the column(s) in ascending order

--sort-descending
: sort the column(s) in descending order

method

WARNING: Any of the sites used by this command may limit the number of queries allowed from a given source in a given period of time and may temporarily block or reject attempts to use their service beyond the quota limit.

utils netblock

Get network CIDR block(s) for IP from WHOIS lookup.

Look up the network address blocks serving the specified IP address(es) using the Python `IPWhois` module.

https://pypi.org/project/ipwhois/

If no arguments are given, the routable address of the host on which `psec` is being run will be determined and used as the default.

```
psec utils netblock [ip [ip ...]]
```

ip
: IP address to use

This command is provided by the python-secrets plugin.

utils set-aws-credentials

Set credentials from saved secrets for use by AWS CLI.

This command directly manipulates the AWS CLI "credentials" INI-style file. The AWS CLI does not support non-interactive manipulation of the credentials file, so this hack is used to do this. Be aware that this might cause some problems (though it shouldn't, since the file is so simple):

```
[default]
aws_access_key_id = [ Harm to Ongoing Matter ]
aws_secret_access_key = [        HOM           ]
```

For simple use cases, you will not need to switch between different users. The default is to use the AWS convention of `default` as seen in the example above. If you do need to support multiple users, the `--user` option will allow you to specify the user.

See also:

- https://aws.amazon.com/cli/
- https://docs.aws.amazon.com/cli/latest/userguide/cli-config-files.html

```
psec utils set-aws-credentials [-U USER]
```

-U <USER>, **--user** <USER>
IAM User who owns credentials

This command is provided by the python-secrets plugin.

utils tfstate backend

Enable Terraform backend support.

Enables the Terraform "backend support" option to move the file `terraform.tfstate` (which can contain many secrets) out of the current working directory and into the current environment directory path.

```
psec utils tfstate backend [--path]
```

--path
Print path and exit

This command is provided by the python-secrets plugin.

utils tfstate output

Retrieve current `terraform output` results.

If the `tfstate` argument is not provided, this command will attempt to search for a `terraform.tfstate` file in (1) the active environment's secrets storage directory (see `environments path`), or (2) the current working directory. The former is documented preferred location for storing this file, since it will contain secrets that *should not* be stored in a source repository directory to avoid potential leaking of those secrets:

```
$ psec environments path
/Users/dittrich/.secrets/psec
```

```
psec utils tfstate output
    [--sort-ascending | --sort-descending]
    [tfstate]
```

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

tfstate
Path to Terraform state file

This command is provided by the python-secrets plugin.

utils yaml-to-json

Convert YAML file(s) to JSON file(s).

You can specify one or more files or directories to convert (including '-' for standard input).

By default the JSON format data will be written to standard output. This is useful for one-off conversion of YAML content to see the resulting JSON, or to produce a file with a different name by redirecting into a new file.

The `--convert` option writes the JSON to a file with the same base name, but with a `.json` extension, then deletes the original YAML file unless the `--keep-original` option is specified. When a directory is passed as an argument with the `--convert` option, *all* files ending in `.yml` in the directory will be processed.

Note: The original format for secrets files and secrets description files was YAML. The format was changed to JSON in a recent release, necessitating that existing secrets descriptions in repositories and/or existing secrets environments be converted. As of now, this utility subcommand provides a mechanism for you to use in making this change. Future releases may include a more user-friendly upgrade mechanism.

Here is a demonstration using an old YAML-style secrets descriptions directory used by tests in the `tests/` subdirectory:

```
$ cp -r tests/secrets /tmp
$ tree /tmp/secrets/
/tmp/secrets/
└── secrets.d
    ├── jenkins.yml
    ├── myapp.yml
    ├── oauth.yml
    └── trident.yml

1 directory, 4 files
$ psec utils yaml-to-json --convert /tmp/secrets/secrets.d
[+] converting '/tmp/secrets/secrets.d/jenkins.yml' to JSON
[+] removing '/tmp/secrets/secrets.d/jenkins.yml'
[+] converting '/tmp/secrets/secrets.d/myapp.yml' to JSON
[+] removing '/tmp/secrets/secrets.d/myapp.yml'
[+] converting '/tmp/secrets/secrets.d/trident.yml' to JSON
[+] removing '/tmp/secrets/secrets.d/trident.yml'
[+] converting '/tmp/secrets/secrets.d/oauth.yml' to JSON
[+] removing '/tmp/secrets/secrets.d/oauth.yml'
$ tree /tmp/secrets/
/tmp/secrets/
└── secrets.d
    ├── jenkins.json
    ├── myapp.json
    ├── oauth.json
    └── trident.json

1 directory, 4 files
```

```
psec utils yaml-to-json [--convert] [--keep-original] [arg [arg ...]]
```

`--convert`
: Convert file(s) in place

`--keep-original`
: Keep original YAML file after conversion

`arg`
: Files and/or directories convert

This command is provided by the python-secrets plugin.

CHAPTER 4

Advanced Usage

4.1 Managing SSH public keys

There is a fundamental problem with using Secure Shell (SSH) access to remote systems, which is dealing with validation of previously unseen host public keys. The vulnerability here, as described by SSH.com, is a MAN-IN-THE-MIDDLE ATTACK.

This problem is bad enough with manually installed computers, either virtual machines or bare-metal servers. In a cloud environment, the problem is exacerbated because every newly instantiated vitual machine may get its own new IP address, domain name, public and private key pairs. To retrieve the public key and/or determine the hashes of the public keys in order to validate them, you must determine the new IP addresses (or DNS names) of every node in the stack in order to remotely log into the instances using SSH, which requires validating the hashes of the SSH public keys… see the problem?

There is a better way, which is to retrieve the public keys and/or fingerprints using the cloud provider's portal console output feature (or the debug output of a program like Terraform). `psec` has sub-commands that parse the console output log to extract the keys and then use Ansible to ensure they are present in the system's `known_hosts` file for all to use immediately. (There is an inverse command to remove these keys when the instance is going to be destroyed and its IP address changed and SSH keys never to be seen again).

This asciicast shows all of the steps involved in instantiating a new cloud instance, extracting and storing its SSH public keys, immediately using SSH without having to validate the key, removing the key, and destroying the instance.

CHAPTER 5

API Reference

5.1 Version

psec.**__version__**
Current python_secrets version.

5.2 psec.google_oauth2

Class for sending cleartext and encrypted emails (optionally with attachments) using OAuth2 authenticated Google SMTP services.

Adapted from:

- https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py
- https://developers.google.com/identity/protocols/OAuth2

See also:

- https://github.com/google/gmail-oauth2-tools/wiki/OAuth2DotPyRunThrough
- http://blog.macuyiko.com/post/2016/how-to-send-html-mails-with-oauth2-and-gmail-in-python.html
- https://developers.google.com/api-client-library/python/guide/aaa_oauth

There are three tasks that can be accomplished using this class:

1. Generating an OAuth2 token with a limited lifetime and a refresh token with an indefinite lifetime to use for login (access_token)
2. Generating a new access token using a refresh token (refresh_token)
3. Generating an OAuth2 string that can be passed to IMAP or SMTP servers to authenticate connections. (generate_oauth2_string())

class psec.google_oauth2.**GoogleSMTP**(*username=None*, *client_id=None*, *client_secret=None*, *refresh_token=None*, *verbose=False*, *gpg_encrypt=False*)

Google OAuth2 SMTP class.

__dict__ = mappingproxy({'__module__': 'psec.google_oauth2', '__doc__': '\n Google O

__init__(*username=None*, *client_id=None*, *client_secret=None*, *refresh_token=None*, *verbose=False*, *gpg_encrypt=False*)

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'psec.google_oauth2'

__weakref__

list of weak references to the object (if defined)

authorize_tokens(*auth_token*)

Return OAuth 2.0 authorization token data following the flow described in "OAuth2 for Installed Applications":

- https://developers.google.com/accounts/docs/OAuth2InstalledApp#handlingtheresponse

Parameters

- **client_id** – Client ID obtained by registering your app.
- **client_secret** – Client secret obtained by registering your app.
- **authorization_code** – code generated by Google Accounts after user grants permission.

Returns The decoded response from the Google Accounts server, as a dict. Expected fields include 'access_token', 'expires_in', and 'refresh_token'.

command_to_url(*command*)

Produce an URL for a given command.

create_msg(*fromaddr*, *toaddr*, *subject*, *text_message=None*, *html_message=None*, *addendum=None*, *encrypt_msg=False*)

Create email message, optionally GPG encrypted.

Parameters

- **fromaddr** – Email `From:` address.
- **toaddr** – Email `To:` address.
- **subject** – Email `Subject:` string.
- **text_message** – Text for body of email message.
- **html_message** – Alternative HTML version of body.
- **addendum** – Signature or other description of the source of the email to be appended to the end of the message following `----`.
- **html_message** – Alternative HTML version of body.

If no alternative HTML is included with a text message body, one will be generated.

If the class was initialized with `gpg_encrypt=True`, the text body will be encrypted with GPG before sending using the key associated with the recipient. If no key is found, or the encryption fails for some other reason, a `RuntimeError` exception is raised.

find_keyid(*recipient*, *keyid=None*)
Locate the GPG keyid for encrypting a message to the recipient.

If a keyid is provided, make sure it matches the recipient and return None if it does not. Otherwise, walk through all keys in the keyring to find a match. If more than one key is found, raise a RuntimeError.

generate_oauth2_string(*base64_encode=False*)
Generates an IMAP OAuth2 authentication string.

See https://developers.google.com/google-apps/gmail/oauth2_overview

Parameters

- **username** – the username (email address) of the account to authenticate
- **access_token** – An OAuth2 access token.
- **base64_encode** – Whether to base64-encode the output.

Returns The SASL argument for the OAuth2 mechanism.

generate_permission_url(*scope='https://mail.google.com/'*)
Generate an OAuth 2.0 authorization URL following the flow described in "OAuth2 for Installed Applications":

- https://developers.google.com/accounts/docs/OAuth2InstalledApp

Parameters

- **client_id** – Client ID obtained by registering your app.
- **scope** – scope for access token, e.g. 'https://mail.google.com'

Returns A URL that the user should visit in their browser.

generate_refresh_token()
Obtains a new OAuth2 authorization token using a refresh token.

See: https://developers.google.com/accounts/docs/OAuth2InstalledApp#refresh

Parameters

- **client_id** – Client ID obtained by registering your app.
- **client_secret** – Client secret obtained by registering your app.
- **refresh_token** – A previously-obtained refresh token.

Returns The decoded response from the Google Accounts server, as a dict. Expected fields include 'access_token', 'expires_in', and 'refresh_token'.

get_authorization()
Get OAuth 2.0 authorization URL.

get_refresh_token()
Get the OAuth 2.0 refresh token.

logger = <Logger psec.google_oauth2 (WARNING)>

refresh_authorization()
Refresh OAuth 2.0 authorization token data.

send_mail(*fromaddr*, *toaddr*, *msg*)
Send email message.

Parameters

- **fromaddr** – Email `From:` address.
- **toaddr** – Email `To:` address.
- **msg** – Already fully-populated `Message` object.

set_client_id(*client_id=None*)
Store the OAuth 2.0 client ID.

set_client_secret(*client_secret=None*)
Store the OAuth 2.0 client secret.

test_imap(*auth_string*)
Authenticates to IMAP with the given auth_string.

Prints a debug trace of the attempted IMAP connection.

Parameters

- **user** – The Gmail username (full email address)
- **auth_string** – A valid OAuth2 string, as returned by generate_oauth2_string(). Must not be base64-encoded, since imaplib does its own base64-encoding.

test_smtp(*auth_string*)
Authenticates to SMTP with the given auth_string.

Parameters

- **user** – The Gmail username (full email address)
- **auth_string** – A valid OAuth2 string, not base64-encoded, as returned by generate_oauth2_string().

url_escape(*text*)
Escape characters in the URL to reduce risk.

url_format_params(*params*)
Format a parameterized URL.

url_unescape(*text*)
Return URL to standard form.

5.3 psec.secrets_environment

Secrets environment class and related variables, functions.

class psec.secrets_environment.**SecretsEnvironment**(*environment=None*, *secrets_basedir=None*, *secrets_file=None*, *create_root=False*, *defer_loading=False*, *export_env_vars=False*, *preserve_existing=False*, *env_var_prefix=None*, *source=None*, *verbose_level=1*)
Class for handling secrets environment metadata.

Provides an interface to the directory contents for a secrets environment, including groups descriptions, a tmp/ directory, and any other required directories.

Typical usage example:

```
from psec.secrets_environment import SecretsEnvironment

se = SecretsEnvironment(environment='env_name')
```

environment
Name of the environment.

secrets_basedir
Base directory path to environment's storage.

secrets_file
File name for storing secrets (defaults to 'secrets.json').

create_root
Controls whether the root directory is created on first use.

defer_loading
Don't load values (just initialize attributes).

export_env_vars
Export all variables to the environment.

preserve_existing
Don't over-write existing environment variables.

env_var_prefix
Prefix to apply to all exported environment variables.

source
Directory path from which to clone a new environment.

verbose_level
Verbosity level (pass from app args).

__dict__ = mappingproxy({'__module__': 'psec.secrets_environment', '__doc__': "\n Cl

__init__(*environment=None*, *secrets_basedir=None*, *secrets_file=None*, *create_root=False*, *defer_loading=False*, *export_env_vars=False*, *preserve_existing=False*, *env_var_prefix=None*, *source=None*, *verbose_level=1*)
Initialize secrets environment object.

__module__ = 'psec.secrets_environment'

__str__()
Produce string representation of environment identifier

__weakref__
list of weak references to the object (if defined)

changed()
Return boolean reflecting changed secrets.

check_duplicates(*data=None*)
Check to see if any 'Variable' dictionary elements in list match any already defined variables. If so, raise RuntimeError().

Parameters **data** – list of dictionaries containing secret descriptions

Returns None

clone_from(*src: Union[pathlib.Path, str]*)
Clone from existing definition file(s)

The source can be (a) a directory full of one or more group descriptions, (b) a single group descriptions file, or (c) an existing environment's descriptions file(s).

delete_secret(*secret*)
Delete a secret and record change.

Parameters **secret** –

type string

Returns

descriptions()

environment_create(*source=None*, *alias=False*, *mode=448*)
Create secrets environment directory

environment_exists(*env=None*, *path_only=False*)
Return whether secrets environment directory exists and contains files other than 'tmp' directory.

find_new_secrets()
Ensure that any new secrets defined in description files are called out and/or become new undefined secrets.
:return:

get_default_value(*variable*)
Return the default value from the Options attribute

get_descriptions_path(*root=None*, *group=None*, *create=False*, *mode=448*)
Return path to secrets descriptions directory or file.

get_environment_path(*env=None*, *subdir=None*, *host=None*)
Returns the absolute path to secrets environment directory or subdirectories within it

get_group(*item*)
Return the group to which an item belongs.

get_groups()
Get the secrets description groups

get_help(*secret*)
Get the help documentation URL for the secret

get_items_from_group(*group*)
Get the variables in a secrets description group

get_options(*secret*)
Get the options for setting the secret

get_prompt(*secret*)
Get the prompt for the secret

get_secret(*secret*, *allow_none=False*)
Get the value of secret

Parameters

- **secret** (*string*) – Name of the secret to get
- **allow_none** (*boolean*) – Allow returning `None`

Returns The value of the secret

Return type string

Raises `SecretNotFoundError` – If value is `None` and `allow_none` is `False`

get_secret_arguments(*variable*)
Get the Arguments of variable from set of secrets descriptions

get_secret_export(*secret*)
Get the specified environment variable for exporting secret

Parameters **secret** –

type string

Returns environment variable for exporting secret

get_secret_type(*variable*)
Get the Type of variable from set of secrets descriptions

get_secrets_basedir(*init=False*, *mode=448*)
Returns the directory path root for secrets storage and definitions.

When more than one environment is being used, a single top-level directory in the user's home directory is the preferred location. This function checks to see if such a directory exists, and if so defaults to that location.

If the environment variable "D2_SECRETS_BASEDIR" is set, that location is used instead.

get_secrets_basename()
Return the basename of the current secrets file

get_secrets_descriptions_dir()
Return the path to the drop-in secrets description directory

get_secrets_file_path(*env=None*)
Returns the absolute path to secrets file

get_tmpdir_path(*create_path=False*)
Return the absolute path to secrets descriptions tmp directory

get_type(*variable*)
Return type for variable or None if no description

is_item_in_group(*item*, *group*)
Return true or false based on item being in group

items()
Return the items from the secrets dictionary.

keys()
Return the keys to the secrets dictionary

logger = <Logger psec.secrets_environment (WARNING)>

read_descriptions(*infile=None*, *group=None*)
Read a secrets group description file and return a dictionary if valid.

Parameters

- **infile** –
- **group** –

Returns dictionary of descriptions

read_secrets(*from_descriptions=False*)
Load the current secrets file.

If no secrets have been set yet and from_descriptions is True, return a dictionary comprised of the keys from the descriptions dictionary defined to be None and set self._changed to ensure these are written out.

read_secrets_and_descriptions(*ignore_errors=False*)
Read secrets descriptions and secrets.

read_secrets_descriptions(*ignore_errors=False*)
Load the descriptions of groups of secrets from a .d directory

requires_environment(*path_only=False*)
Provide consistent error handling for any commands that require an environment actually exist in order to work properly.

secrets_basedir_exists()
Return whether secrets root directory exists

secrets_file_path_exists()
Return whether secrets file exists

set_secret(*secret*, *value=None*)
Set secret to value and record change

Parameters

- **secret** –

  type string

- **value** –

  type string

Returns

verbose_level
Returns the verbosity level.

write_descriptions(*data={}*, *group=None*, *mode=448*, *mirror_to=None*)
Write out the secrets descriptions to a file.

write_secrets()
Write out the current secrets if any changes were made

psec.secrets_environment.**generate_secret**(*secret_type*, ***kwargs*)
Generate secret of the specified type.

psec.secrets_environment.**is_generable**(*secret_type=None*)
Return boolean for generability of this secret type.

5.4 psec.utils

Utility functions.

Author: Dave Dittrich <dave.dittrich@gmail.com>

URL: https://python_secrets.readthedocs.org.

class psec.utils.**CustomFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)
Custom class to control arparse help output formatting.

__module__ = 'psec.utils'

class psec.utils.**Memoize**(*fn*)
Memoize(fn) - an instance which acts like fn but memoizes its arguments.

Will only work on functions with non-mutable arguments. Hacked to assume that argument to function is whether to cache or not, allowing all secrets of a given type to be set to the same value.

__call__(**args*)
Call self as a function.

__dict__ = mappingproxy({'__module__': 'psec.utils', '__doc__': 'Memoize(fn) - an in

__init__(*fn*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'psec.utils'

__weakref__
list of weak references to the object (if defined)

class psec.utils.**Timer**(*task_description='elapsed time'*, *verbose=False*)
Timer object usable as a context manager, or for manual timing.

Based on code from http://coreygoldberg.blogspot.com/2012/06/python-timer-class-context-manager-for.html # noqa

As a context manager, do:

from timer import Timer

url = 'https://github.com/timeline.json'

with Timer() as t: r = requests.get(url)

print 'fetched %r in %.2f millisecs' % (url, t.elapsed*1000)

__dict__ = mappingproxy({'__module__': 'psec.utils', '__doc__': "\n Timer object usa

__enter__()
Record initial time.

__exit__(**args*)
Record final time.

__init__(*task_description='elapsed time'*, *verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'psec.utils'

__weakref__
list of weak references to the object (if defined)

elapsed(*start='__enter__'*, *end='__exit__'*)
Return a formatted string with elapsed time between 'start' and 'end' kwargs (if specified) in HH:MM:SS.SS format.

elapsed_raw(*start='__enter__'*, *end='__exit__'*)
Return the elapsed time as a raw value.

get_lap(*lap='__exit__'*)
Get the timer for label specified by 'lap'

lap(*lap='__lap__'*)
Records a lap time. If no lap label is specified, a single 'last lap' counter will be (re)used. To keep track of more laps, provide labels yourself.

start(*lap=None*)
Record starting time.

stop()
Record stop time.

psec.utils.**atree**(*dir*, *print_files=True*, *outfile=None*)
Produces the tree structure for the path specified on the command line. If output is specified (e.g., as sys.stdout) it will be used, otherwise a list of strings is returned.

Uses anytree: https://anytree.readthedocs.io/en/latest/

Parameters

- **dir** –
- **print_files** –
- **outfile** –

Returns str

psec.utils.**bell**()
Output an ASCII BEL character to `stderr`.

psec.utils.**clear_saved_default_environment**(*cwd=None*)
Remove saved default environment file.

psec.utils.**copyanything**(*src*, *dst*)
Copy anything from src to dst.

psec.utils.**copydescriptions**(*src: pathlib.Path*, *dst: pathlib.Path*)
Just copy the descriptions portion of an environment directory from src to dst.

psec.utils.**ensure_secrets_basedir**(*secrets_basedir=None*, *allow_create=False*, *allow_prompt=False*, *verbose_level=1*)
Ensure that the secrets basedir exists.

If the path is within the user's home directory, it is OK to create the directory automatically if it does not exist. This was the original behavior. If the path does exist and contains file, but does not have the special marker, that will be considered an error the user needs to resolve.

For paths that lie outside the user's home directory, the user must explicitly confirm that it is OK to create the directory by responding to prompts (when possible) or by using the *–init* option flag or *psec init* command.

psec.utils.**find**(*lst*, *key*, *value*)
Searches a list of dictionaries by value of a specified key.

Find the first item from a list of dicts where the key identified by `key` has the value specified by `value`.

Parameters

- **lst** (*list of dict*) – List of dictionaries to search
- **key** (*str*) – Key to compare
- **value** (*str*) – Value to find

Returns Index to the first entry with the matching value or `None`

psec.utils.**get_default_environment**(*cwd=None*)
Return the default environment identifier.

There are multiple ways for a user to specify the environment to use for python_secrets commands. Some of these involve explicit settings (e.g., via command line option, a saved value in the current working directory, or an environment variable) or implicitly from the name of the current working directory.

psec.utils.**get_default_secrets_basedir**()
Return the default secrets base directory path.

psec.utils.**get_environment_paths**(*basedir=None*)
Return sorted list of valid environment paths found in *basedir*.

psec.utils.**get_files_from_path**(*path=None*)
Gets a list of absolute paths to one or more files associated with a path.

If `path` is a directory, the files contained in it are returned, otherwise the path to the file is the only item in the list.

Parameters **path** (*str*) – Candidate path.

Returns A list of one or more absolute file paths.

Return type list

psec.utils.**get_fs_type**(*mypath*)
Identifies the file system type for a specific mount path.

Parameters **mypath** (*str*) – Candidate path.

Returns File system type for partition containing `mypath`.

Return type string

psec.utils.**get_local_default_file**(*cwd=None*)
Returns the path to the local identifier file.

psec.utils.**get_myip**(*method='random'*)
Return current routable source IP address.

psec.utils.**get_myip_methods**(*include_random=False*)
Return list of available method ids for getting IP address.

psec.utils.**get_netblock**(*ip=None*)
Derives the CIDR netblocks for an IP via WHOIS lookup.

Parameters **ip** (*str*) – IP address

Returns One or more CIDR blocks

Return type string

psec.utils.**get_output**(*cmd=['echo', 'NO COMMAND SPECIFIED'], cwd='/home/docs/checkouts/readthedocs.org/user_builds/python-secrets/checkouts/stable/docs', stderr=-2, shell=False*)
Uses `subprocess.check_ouput()` to run a sub-command.

Parameters

- **cmd** (*list*) – Argument list
- **cwd** (*str*) – Directory to use for current working directory by shell
- **stderr** (*file handle*) – Where should `stderr` be directed? (default: `subprocess.STDOUT`)
- **shell** (*bool*) – Use a shell (default: `FALSE`)

Returns Output from command

Return type list of str

psec.utils.**get_saved_default_environment**(*cwd=None*)
Return environment ID value saved in local file or None.

psec.utils.**getmount**(*mypath*)
Identifies the filesystem mount point for the partition containing `mypath`.

Parameters **mypath** (*str*) – Candidate path.

Returns The mount point for the filesystem partition containing `path`.

Return type string

psec.utils.**getmount_fstype**(*mypath*)
Identifies the file system type for a specific mount path.

Parameters **mypath** (*str*) – Candidate path.

Returns File system type for partition containing `mypath`.

Return type string

psec.utils.**is_secrets_basedir**(*basedir=None*, *raise_exception=True*)
Validate secrets base directory by presence of a marker file.

Returns False if the directory either does not exist or does not contain the expected marker file, or True otherwise.

psec.utils.**is_valid_environment**(*env_path*, *verbose_level=1*)
Check to see if this looks like a valid environment directory.

Parameters

- **env_path** – Path to candidate directory to test.
- **verbose_level** – Verbosity level (pass from app args)

Returns A boolean indicating whether the directory appears to be a valid environment directory or not based on contents including a 'secrets.json' file or a 'secrets.d' directory.

psec.utils.**myip_http**(*arg=None*)
Use an HTTP service that only returns IP address.

psec.utils.**myip_resolver**(*arg=None*)
Use DNS resolver to get IP address.

psec.utils.**natural_number**(*value*)
Tests for a natural number.

Parameters **value** – The value to test

Returns A boolean indicating whether the value is a natural number or not.

psec.utils.**permissions_check**(*basedir='.'*, *verbose_level=0*)
Check for presense of pernicious overly-permissive permissions.

psec.utils.**prompt_options_dict**(*options=None*, *by_descr=True*, *prompt='Select from the following options'*)
Prompt the user for a string using option dictionaries.

These dictionaries map a descriptive name to an identifier:

```
{'descr': 'DigitalOcean', 'ident': 'digitalocean'}
```

psec.utils.**prompt_options_list**(*options=None*, *default=None*, *prompt='Select from the following options'*)
Prompt the user for a string using a list of options.

The options will be one of the following:

'*' - Any user input 'A,*' - 'A', or any user input. 'A,B' - Only choices are 'A' or 'B'.

psec.utils.**prompt_string**(*prompt='Enter a value'*, *default=None*)
Prompt the user for a string and return it

psec.utils.**redact**(*string*, *redact=False*)

psec.utils.**remove_other_perms**(*dst*)
Make all files in path `dst` have `o-rwx` permissions.

NOTE: This does not work on file system types `NTFS`, `FAT`, or `FAT32`. A log message will be produced when this is encountered.

psec.utils.**require_options**(*options*, **args*)

psec.utils.**safe_delete_file**(*file_name=None*, *passes=3*, *verbose=False*)

psec.utils.**save_default_environment**(*environment=None*, *cwd=None*)
Save environment identifier to local file for defaulting.

psec.utils.**secrets_basedir_create**(*basedir=None*, *mode=448*)
Create secrets root directory

psec.utils.**secrets_tree**(*env=None*, *outfile=None*)
Produces the tree structure for groups and secrets in an environment.

If output is specified (e.g., as sys.stdout) it will be used, otherwise a list of strings is returned.

Uses anytree: https://anytree.readthedocs.io/en/latest/

Parameters

- **environment_dir** –
- **outfile** –

Returns str

psec.utils.**show_current_value**(*variable=None*)
Pretty-print environment variable (if set).

psec.utils.**umask**(*value*)
Set umask.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at this repository's GitHub issues page (https://github.com/davedittrich/python_secrets/issues).[1]

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Output using the `--debug` and `-vvv` flags.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues[1] for bugs. Anything tagged with **bug** is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues[1] for features. Anything tagged with **feature** is open to whoever wants to implement it.

[1] https://github.com/davedittrich/python_secrets/issues

6.1.4 Write Documentation

python_secrets, like pretty much every open source project, could always use more user-friendly documentation. That includes this official python_secrets documentation, docstrings in source code, and around the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/davedittrich/python_secrets/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement the feature.
- Remember that this is a volunteer-driven project, and that contributions (i.e., pull requests) *are always welcome*. ;)

6.2 Get Started!

Ready to contribute? Here's how to set up *python_secrets* for local development.

1. Fork the *python_secrets* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python_secrets.git
```

3. Ensure Bats is ready to use for testing. Bats assertion libraries are assumed to be installed in Git cloned repositories at the same directory level as the python_secrets repository:

```
$ git clone https://github.com/ztombol/bats-support.git
$ git clone https://github.com/jasonkarns/bats-assert-1.git
```

4. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv python_secrets
$ cd python_secrets/
$ python setup.py develop
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and bandit (security) tests, including testing other Python versions with tox:

```
$ make test
```

   To get flake8 and tox, just python -m pip install them into your virtualenv.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list of changes in `HISTORY.rst` and documentation on use in `README.rst`, `docs/usage.rst`, and `parser.epilog` for CLI commands.
3. The pull request should work for the versions of Python defined in `tox.ini` and `.travis.yml`. Check https://travis-ci.org/davedittrich/python_secrets/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of Python unit tests:

```
$ python -m unittest tests.test_secrets
```

To run a subset of Bats tests:

```
$ bats tests/secrets.bats
```

CHAPTER 7

Credits

7.1 Development Lead

- Dave Dittrich <dave.dittrich@gmail.com>

7.2 Contributors

None yet. Why not be the first?

7.3 Support

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-pypackage

Development of this program was supported in part under an Open Source Development Grant from the Comcast Innovation Fund.

CHAPTER 8

History

8.1 23.4.2 (2023-04-20)

8.1.1 Added

- Added bats test for *secrets find* command.

8.1.2 Changed

- Switched to using f-strings for formatting.
- Fixed broken bats tests.
- Update GitHub Actions workflows.

8.2 23.4.1 (2023-04-19)

8.2.1 Added

- Added *secrets find* command.
- Added support for new variable type *boolean*.

8.2.2 Changed

- Updated GitHub Actions workflows (default to Python 3.9.16).
- Drop Python 3.7, 3.8, add Python 3.11 (default to 3.10) for *tox*.
- Fixed downstream dependency and *pip* installation problems.

- Resolved new *pep8* and *bandit* findings.

8.3 22.6.1 (2022-06-21)

8.3.1 Added

- Added *–ignore-missing* option to continue when settings variables.
- Added 'Operational Security' section to README.

8.4 22.6.0 (2022-06-10)

8.4.1 Added

- Add *about* command to expose selected settings for situational awareness.
- Add *pytest* code coverage reporting.
- Add BATS runtime tests related to changes.

8.4.2 Changed

- Fix caching bug with non-unique secret generation.
- Fix bugs with setting/deleting secrets.
- Improve secrets basedir initialization logic.
- Expand use of *pathlib.Path*.
- Improvements to source code, test, and vscode launch configuration quality.

8.5 22.5.1 (2022-05-25)

8.5.1 Changed

- Switch to using factory pattern for secrets generation.
- General code quality and test improvements.
- Improve *secrets get* command logic and help.
- Fix *utils yaml-to-json* subcommand and tests.
- Resolve setuptools warnings.
- Separate utility functions from *utils* subcommands.

8.5.2 Removed

- Retire *consul_key* secret type in favor of *token_base64*.
- Retire insecure secrets types (e.g., use of SHA1).

8.6 22.5.0 (2022-05-11)

8.6.1 Added

- Test support for Python 3.10.
- Add better logging controls.

8.6.2 Changed

- Generalize Google OAuth2 email functionality.
- Improve use and testing of exceptions.

8.7 22.1.0 (2022-01-22)

8.7.1 Added

- Add *init* command and *–init* flag to initialize secrets base directory.
- Ensure overridden values via flags are exported to process environment for subprocesses to use.
- Add missing tests for features added in a previous release.
- Add and start using application-specific exception classes.

8.7.2 Changed

- Move functions and variables to *utils* to improve reuse ability.
- Use *get_* prefix more consistently for getter method/function names.
- Over-ride cliff formatter class globally in app parser setup.
- Use *pathlib.Path* for paths for cleaner code.
- Fix bugs in *environments delete* command.
- Fix bugs in *–from-options* feature of *secrets get* and *secrets set*.
- Improvements to source code, test, and vscode launch configuration quality.

8.8 21.11.0 (2021-11-22)

8.8.1 Added

- Add *Help* attribute to descriptions for URL to more information.

8.8.2 Changed

- General code quality, documentation, and testing enhancements
- Move *tmpdir* path creation to *secrets_environment.SecretsEnvironment()*.
- Move *umask()* function and variables to *utils*.

8.8.3 Removed

- Drop Python 3.6 support due to it being EOL.

8.9 21.9.1 (2021-09-15)

8.9.1 Added

- Added *secrets tree* subcommand.

8.9.2 Changed

- Fixed bugs with *environments path –tmpdir* subcommand and *run* subcommand with *–elapsed* option when no environment exists.
- Changed license file name.
- Improved documentation.

8.10 21.9.0 (2021-09-07)

8.10.1 Added

- Increased test coverage to address bugs (below) being fixed.

8.10.2 Changed

- Fixed bugs in *Makefile* and *tox.ini* file.
- Fixed bug setting undefined variables.
- Switched from *numpy* to Python *secrets* module for random bytes.
- Increased key size from 16 to 32 bits for *consul_key*, *token_hex* and *token_urlsafe*.

8.11 21.8.0 (2021-08-12)

8.11.1 Changed

- Fixed bug in setup.py+setup.cfg

8.12 21.7.0 (2021-07-30)

8.12.1 Added

- Secrets descriptions for demoing HypriotOS Flash mods Medium article

8.12.2 Changed

- Improve *secrets set –from-options*
- General code quality, documentation, and testing enhancements

8.13 21.6.0 (2021-06-23)

8.13.1 Added

- Ability to set and generate secrets from defaults options
- Ability to create an alias for an existing environment
- Allow retroactive mirroring of new secrets

8.13.2 Changed

- Switched from *pbr* to *setuptools_scm* for version numbering
- Switched to more secure random number generation

8.14 21.2.0 (20201-02-23)

8.14.1 Added

- Improve GitHub Actions workflows
- Overall documentation and code enhancements
- Improve handling of wildcards in options list

8.14.2 Changed

- Fix bugs with handling empty lists, cloning environments, BATS tests
- Increase password complexity a bit more
- Fix ReadTheDocs

8.15 20.11.0 (2020-11-17)

8.15.1 Added

- Add *secrets create* and *secrets delete* commands

8.15.2 Changed

- Normalize all logger and exception output text
- Refactoring code for better modulatiry
- Normalize *group create* and *group delete* code
- Normalize *secrets show* and *secrets describe* code
- Fix bug that left variables missing after cloning
- Add Python 3.9 to testing matrix
- Switch from .yml to .json format for secrets
- Expand IP address support in *utils* subcommand

8.16 20.8.1 (2020-08-11)

8.16.1 Changed

- Fixes to v20.8.0

8.17 20.8.0 (2020-08-11)

8.17.1 Added

- Add GitHub workflow to publish to test.pypi.org
- Add *secrets backup* and *secrets restore* logic
- Open web browser to documentation for help

8.17.2 Changed

- Go back to date-based version numbering
- General CI/CD workflow updates
- Improve directory handling in *environments path*

8.18 20.2.15 (2012-02-15)

8.18.1 Added

- Added Python 3.8 support to test matrix

8.18.2 Changed

- Fix bug in *environments default*
- Put elapsed time (and BELL) on stdout
- Fix bug in *environments tree*
- Allow setting vars using diff names+environment

8.19 19.12.0 (2019-12-16)

8.19.1 Added

- Add and document new *boolean* data type
- Add *groups delete* command

8.19.2 Changed

- Improve default environment handling
- Improve tox+BATS testing
- Address security issue per "Your xkcd passwords are pwned" article
- General code quality and test improvements
- Add protection from over-writing existing env vars
- Add *Options* attribute

8.20 19.11.1 (2019-11-29)

8.20.1 Changed

- Enhancements to better support Windows 10
- Allow cloning group descriptions from environment
- Fix tty/no-tty handling with *environments delete*
- Expose terraform command on *-v*
- Validate variable exists in environment
- Fix broken *environments tree* code

8.21 19.10.1 (2019-10-20)

8.21.1 Changed

- Move BATS unit tests into tox testing
- Avoid attempting interactive things when no tty
- Improve file and directory permissions logic

8.22 19.10.0 (2019-10-14)

8.22.1 Added

- Working SSH key and configuration management
- Use *bullet* for interactive list selection
- Elapsed timer feature
- Parsing of terraform output to extract SSH public keys
- *umask* control for better new file permission settings
- Support configuring terraform *tfstate* backend
- Allow setting secrets by copying from another environment

8.22.2 Changed

- Numerous bug fixes
- Refine testing
- Option to only show undefined variables
- Sort environments when listing

8.23 19.9.0 (2019-09-05)

8.23.1 Added

- Add *environments delete* subcommand
- Allow cloning environment from an existing one

8.23.2 Changed

8.24 19.8.3 (2019-08-28)

8.24.1 Changed

- Dynamically get version number
- General testing enhancements
- General code quality enhancements
- Ensure more secure file permissions

8.25 19.8.2 (2019-08-23)

8.25.1 Changed

- General code quality enhancements

8.26 19.8.0 (2019-08-22)

8.26.1 Added

- IP address determination
- Allow cloning new group in an empty environment
- Make *python -m psec* work
- JSON output method
- Environment aliasing feature

8.26.2 Changed

- General code quality and testing enhancements
- Be more explicit about default environment
- Tighten permissions on cloned environments/groups
- Add insecure permissions checking

8.27 19.5.1 (2019-05-08)

8.27.1 Changed

Add *HISTORY.rst* file

8.28 19.4.5 (2019-05-08)

8.28.1 Added

- Add command *ssh config* to manage SSH configuration snippet for use by *update-dotdee* to generate ~/.ssh/config file
- Add command *ssh known-hosts add* and *ssh known-hosts remove* to manage system known_hosts file(s)

8.28.2 Changed

- Generalized exception to fix –version bug
- Clean up temporary docs/psec_help.txt file

8.29 19.4.4 (2019-04-21)

8.29.1 Changed

- Fix Bats dependencies/tests
- Fix broken documentation (wt?)
- Fix messed up release tagging

8.30 19.4.0 (2019-04-19)

8.30.1 Added

- Python 3.7 coverage for Travis CI

8.30.2 Changed

- Complete –help output (epilog text) in all commands
- Install a script 'psec' to complement console_script entry point
- Clarify arguments in –help output

8.30.3 Deprecated

- The 'python_secrets' command is now just 'psec'

8.31 19.3.1 (2019-04-06)

8.31.1 Added

- Add `environments rename` command
- Add `utils set-aws-credentials` command to mirror AWS CLI credentials
- Use `autoprogram_cliff` for self-documentation
- Add `cliff.sphinxext` for documentation

8.31.2 Changed

- Refactored `SecretsEnvironment()` so `autoprogram_cliff` works

8.32 18.11.0 (2018-11-09)

8.32.1 Added

- Add "–type" option to "secrets describe"
- Improve visibility into default environment
- Add screencasts to documenation
- Add RST checks to ensure PyPi documentation works
- Add feedback about minimum Python version
- Add `--json` output to `environments path`
- Add reference to proof-of-concept using goSecure fork

8.32.2 Changed

- The "secrets describe" command now describes variables and types
- Allow `secrets set` to set any type (not just `string`)

8.33 18.9.0 (2018-09-27)

8.33.1 Added

- Switched to calendar version numbering
- Finish GPG encrypted email delivery of secrets
- `groups create` command
- Improve error handling consistency when no environment exists

8.34 0.16.0 (2018-09-12)

8.34.1 Added

- Use attribute maps instead of lookup loops
- Add Prompt attribute in descriptions for better UX when setting variables
- Note new undefined variables when adding groups or `environments create --clone-from`
- When exporting vars, also export PYTHON_SECRETS_ENVIRONMENT w/environment name
- Add reference to Python Security coding information
- `environments tree` command
- `environments path` command with features supporting Ansible Lookup Plugin
- `secrets get` command
- `groups path` command
- `environments default` command

8.35 0.14.0 (2018-08-30)

8.35.1 Added

- Option to export secrets as environment variables (with optional prefix)
- Can now set secrets (any specified or all undefined) via command line
- `utils myip` command returns routable IP address (with CIDR option)
- `run` command allows running commands with exported environment variables

8.35.2 Changed

- Renamed `template` comamnd to `utils tfoutput`

8.35.3 Removed

- Dropped support for Python 3.4, 3.5, since `secrets` module only in Python >= 3.6

8.36 0.10.0 (2018-08-23)

8.36.1 Added

- New `string` type for manually set secrets
- `secrets path` command provides path to secrets `.yml` file
- `template` command (Jinja templating)
- Default environment to basename of cwd

- Clone environment from skeleton directory in repo

8.37 0.9.1 (2018-08-19)

8.37.1 Added

- `secrets describe` command
- `environments create` command
- `environments list` command
- Expand secrets types and generation methods
- Add initial feature for sending secrets via email using Google OAuth2 SMTP

8.37.2 Removed

- Drop Python 2.7 support (at least for now...)

8.37.3 Security

- Add `six` for securing `input` call

8.38 0.8.0 (2018-05-11)

(TBD)

8.39 0.4.0 (2018-05-01)

(TBD)

8.40 0.3.6 (2018-04-29)

(TBD)

8.41 0.3.0 (2018-04-27)

- First release on PyPI.

CHAPTER 9

License

```
Copyright (c) 2018--2021 Dave Dittrich <dave.dittrich@gmail.com>.
All rights reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

S

T

V