Envie Documentation

Release 0.4.37-dev

Radomir Stevanovic

Dec 04, 2017

Contents:

CHAPTER 1

Quick start

Start by installing Envie. You can install it system-wide (for all users) with:

```
# system-wide install
sudo pip install envie
```

or, if you prefer keeping it user-local, or just trying it out from the source without installing, please take a look at *Install* instructions.

After installing, configuring is recommended, *but not required*. You can run a quick interactive config with:

```
# short step-by-step interactive configuration
envie config
```

At bare minimum, grant yourself at least bash completions and better experience by *registering Envie* (sourcing it in your `.bashrc`). For details, see *Configure*.

After Envie is configured, open a new shell, or simply source your `.bashrc`:

```
. ~/.bashrc
```

1.1 Start with `envie help`

```
Your virtual environments wrangler. Holds no assumptions on virtual env dir
location in relation to code, but works best if they're near (nested or in level).

Usage:
    envie [OPTIONS] [DIR] [KEYWORDS]
    envie SCRIPT
    envie {create [ENV] | remove | list [DIR] [KEYWORDS] | find [DIR] [KEYWORDS] |
           python [SCRIPT] | run CMD | config | index | help | --help | --version}

Commands:
    python SCRIPT  run Python SCRIPT in the closest environment
    run CMD        execute CMD in the closest environment. CMD can be a
                   script file, command, builtin, alias, or a function.

    create [ENV]   create a new virtual environment (alias for mkenv)
    remove         destroy the active environment (alias for rmenv)

    list [DIR]     list virtual envs under DIR (alias for lsenv)
    find [DIR]     like 'list', but also look above, until env found (alias for␣
↪findenv)

    config         interactively configure Envie
    index          (re-)index virtualenvs under custom basedir (default: $HOME)
    --help, help   this help
    --version      version info

The first form is basically an alias for 'chenv -v [DIR] [KEYWORDS]'. It interactively
activates the closest environment (relative to DIR, or cwd, filtered by KEYWORDS).
If a single closest environment is detected, it is auto-activated.

The second form is a shorthand for executing python scripts in the closest
virtual environment, without the need for a manual env activation. It's convenient
for hash bangs:
    #!/usr/bin/env envie
    # Python script here will be executed in the closest virtual env

The third form exposes explicit commands for virtual env creation, removal, discovery,
↪ etc.
For more details on a specific command, see its help with '-h', e.g. 'envie find -h'.
Each of these commands has a shorter alias (mkenv, lsenv, findenv, chenv, rmenv, etc).

Examples:
    envie python               # run interactive Python shell in the closest env
    envie manage.py shell      # run Django shell in the project env (auto activate)
    envie run /path/to/exec    # execute an executable in the closest env
    envie ~ my cool project    # activate the env with words my,cool,project in its␣
↪path,
                               # residing somewhere under your home dir (~)
    mkenv -3r dev-requirements.txt devenv    # create Python 3 virtual env in ./
↪devenv and
                                             # install pip packages from dev-
↪requirements.txt
    mkenv -ta && pytest && rmenv -f          # run tests in a throw-away env with␣
↪packages
                                             # from the closest 'requirements.txt'␣
↪file
```

Detailed *commands reference* is available.

CHAPTER 2

Setup

2.1 Install

For convenience, Envie is packaged and distributed via PyPI as a Python package named `envie`. Full source code is available on GitHub.

You can install Envie in several ways.

2.1.1 1. System-wide install via pip

The simplest and in most cases the recommended way of installing Envie is via pip global install:

```
sudo pip install envie
```

All executable Envie scripts (`envie`, `envie-tmp` and `envie-tools`) will be installed in system `/usr/local/bin/` directory and will be available to all users of the system.

Tip: You can check if Envie is properly installed with:

```
$ envie --version
Envie 0.4.33 command from /usr/local/bin/envie
```

Actually, with this command you can also check if Envie is being run as a **command**, or as a **function**. Almost always you want `envie` to be a function – otherwise you won't be able to easily activate virtual environments discovered. See *Configure*.

2.1.2 2. User-local install via pip

To install to the Python **user install** directory (typically `~/.local`):

```
pip install --user envie
```

This is as a good option if you do not wish to (or can not) install Envie for all users. Executable scripts will be located in your `$HOME/.local/bin/` directory.

If you're not already using other CLI tools installed this way, you'll have to configure your `PATH` to make `envie` executable accessible. Add this to your `~/.bashrc`:

```
export PATH="$PATH:$HOME/.local/bin"
```

2.1.3 3. Manual install from source

Clone with `git`:

```
git clone https://github.com/randomir/envie.git ~/Downloads/envie-master
```

or download a zip archive.

After cloning/downloading, you have to either:

1. **Source** `scripts/envie`, like this:

   ```
   . ~/Downloads/envie-master/scripts/envie
   ```

   Now you'll be running Envie **as a function**, check it out:

   ```
   $ envie --version
   Envie 0.4.33 function from /home/stevie/Downloads/envie-master/scripts/envie
   ```

   To ensure `envie` function is always available, add the sourcing statement to your `.bashrc`, or simply run:

   ```
   envie config --register
   ```

   Also, be sure to check how to *Configure* other aspects of Envie.

or

2. **Symlink** `scripts/envie` executable to your (local) bin directory, for example:

   ```
   ln -s ~/Downloads/envie-master/scripts/envie ~/bin/envie
   ```

   This assumes your `PATH` already includes `~/bin/`. If not, add it just like above, by appending `export PATH="$PATH:$HOME/bin"` to your `~/.bashrc`.

Important: When manually installing Envie as a **command** – and not a function, symlinking `envie` executable to a `PATH`-discoverable location is a MUST. Otherwise Envie command **will not** function properly.

The reason you have to symlink, and not just copy is, ultimately, cross-platform support and *fuzzy environment name filtering*. Namely, cross-platform implementation of some basic tools (GNU `readlink`, `realpath`) and fuzzy-filtering is provided via Python package `envie` (module `envie.filters`). When Envie is pip-installed, this package is available – but when running from source, Envie has to be able to locate it (relative to the `envie` executable).

Hint: An important exception to the symlinking note above is when you know you'll be running Envie **only as a function**, never as a command (*Hint: you probably only want it as a function*).

2.2 Configure

Envie configuration is stored in a config file: `$HOME/.config/envie/envierc`, as a series of shell variables assignments. It is read once per sourcing or execution. In the default setup (when Envie is sourced from `.bashrc`), that means the configuration is read once per Bash session.

Configuration file can be (re-)generated with a guided quick-start script:

```
envie config
```

If you are installing/configuring Envie on a dev machine, you're probably safe to answer all questions with the default (pressing `Enter`):

```
Add to ~/.bashrc (strongly recommended) [Y/n]?
Use locate/updatedb for faster search [Y/n]?
Common ancestor dir of all environments to be indexed [/home/stevie]:
Update index periodically (every 15min) [Y/n]?
Refresh stale index before each search [Y/n]?

Envie added to /home/stevie/.bashrc.
Config file written to /home/stevie/.config/envie/envierc.
Crontab updated.
Indexing environments in '/home/stevie'...Done.
```

2.2.1 Find vs. Locate (aka The faster search)

By default, `envie` uses the `find` command to search for environments. That approach is pretty fast when searching shallow trees. However, if you have deeper directory trees, it's often faster to use a pre-built directory index (i.e. the `locate` command). To enable a combined `locate/find` approach to search, you must run `envie config` and answer *Yes* when asked about the `locate/updatedb` usage.

Tip: In a production/server environment, you might not want to use *locate* method and run cron updatedb jobs every 15min.

Actually, you **can** still use *locate*, but rebuild the index manually with `envie index` (when deemed necessary), or instruct Envie to *"refresh stale index before each search"*.

Note: When `locate` is enabled (and index built), the combined approach is used by default (if not overriden with `-f` or `-l` switches). In the combined approach, if `find` doesn't finish within 400ms, search via `find` is aborted and `locate` is allowed to finish (faster).

2.2.2 The unconfigured mode

When you run Envie without explicitly configuring it, a set of safe *defaults* will be used. Most notably, only `find` method will be used for environments discovery.

2.2.3 Sourcing vs. Executing

Envie can be run directly (by executing `envie` script), or as a shell function (which is defined when `envie` script is sourced).

Either way you chose to run Envie, it will behave the same – with one notable exception:

Warning: If Envie is **not run as a function**, it will **not be able to activate a virtual environment**.

The effects of this will be visible in two scenarios:

`envie create/mkenv` Environment will be created, and requirements/packages will be installed, but virtualenv will not be activated in your current shell.

`envie/chenv` Environments will be listed/selected, but it will not be activated in the current shell.

2.2.4 A reasonable minimum

However you decide on *locate* and crontab index updating, the simplest fully functional (bash completions included) and minimal-performance-overhead configuration is achieved with:

```
envie config --register
```

This will add Envie sourcing statement to your `.bashrc` and ensure you have a working `envie` function, along with the accompanying shorthand aliases like `mkenv`, `lsenv`, etc.

2.2.5 The defaults

`cat $HOME/.config/envie/envierc`:

```
_ENVIE_DEFAULT_ENVNAME="env"
_ENVIE_DEFAULT_PYTHON="python"
_ENVIE_CONFIG_DIR="$HOME/.config/envie"
_ENVIE_USE_DB="1"
_ENVIE_DB_PATH="$HOME/.config/envie/locate.db"
_ENVIE_INDEX_ROOT="$HOME"
_ENVIE_CRON_INDEX="1"
_ENVIE_CRON_PERIOD_MIN="15"
_ENVIE_LS_INDEX="1"
_ENVIE_FIND_LIMIT_SEC="0.4"
_ENVIE_LOCATE_LIMIT_SEC="4"
_ENVIE_UUID="28d0b2c7bc5245d5b1278015abc3f0cd"
```

2.2.6 Config variables

`_ENVIE_DEFAULT_ENVNAME` Name of the virtual environment base directory. The usual values are: `env`, `.env`, `.venv`, and `pythonenv`.

`_ENVIE_DEFAULT_PYTHON` Preferred Python interpreter. Use something like `python` (the system default), `python3` (the default version of Python 3), or `/usr/local/bin/python3.6`.

`_ENVIE_USE_DB` Should Envie use `locate`/`updatedb` when looking for virtual environments on disk? (boolean: `0`/`1`). Defaults to yes, but in server environments you may be inclined to fall-back to `find`-only approach. Please note you still may use the `locate` approach with manual or on-demand indexing.

_ENVIE_DB_PATH Path to Envie's local `updatedb` database.

_ENVIE_INDEX_ROOT Root dir for `updatedb` index. Set it to a common ancestor of all virtual environments **you wish to index**. Defaults to `$HOME`, but you may want to set it to something like `/srv`, `/var/www`, or even `/`. Note that this setting does not affect the `find` search.

_ENVIE_CRON_INDEX Should Envie refresh its `updatedb` database with a periodic cron job? (boolean: `0`/`1`). If the appropriate question during `envie config` is answered affirmatively, an user-local cron job is added with `crontab`.

_ENVIE_CRON_PERIOD_MIN Database refresh period (1-60 minutes).

_ENVIE_LS_INDEX Should Envie initiate `updatedb` upon each environment search with `lsenv`/`envie list`/`findenv`/`envie find`/`chenv`/`envie` if the index is older than `_ENVIE_LOCATE_LIMIT_SEC` seconds? (boolean: `0`/`1`).

_ENVIE_FIND_LIMIT_SEC Limit in seconds on execution time for `find` when searching for environments, if a locate database is used.

_ENVIE_LOCATE_LIMIT_SEC Max. allowed age for locate database, in seconds. If database is older than this, index rebuild is called if `_ENVIE_LS_INDEX=1`, or a warning message is displayed otherwise.

CHAPTER 3

Commands Reference

3.1 Calling Envie

The `envie` script (or a *shell function*) has three calling forms – two “shortcut” forms, and general/all-purpose form:

3.1.1 1. find & activate

```
envie [OPTIONS] [DIR] [KEYWORDS]
```

The first form interactively activates the closest environment (relative to `DIR`, or the current working directory, filtered by `KEYWORDS`). If a single closest environment is detected, it is auto-activated. This calling form is basically an alias for `chenv -v [DIR] [KEYWORDS]`. For options and details on environments discovery/selection, see *chenv* below.

3.1.2 2. run python script

```
envie SCRIPT
```

The second form is a shorthand for executing python scripts in the closest virtual environment, without the need for a manual env activation. It’s identical in behaviour to `envie python SCRIPT` (see *below*), but more convenient for a hash bang use:

```
#!/usr/bin/env envie
# Python script here will be executed in the closest virtual env
```

3.1.3 3. general

```
envie {create [ENV] | remove |
       list [DIR] [KEYWORDS] | find [DIR] [KEYWORDS] |
       python [SCRIPT] | run CMD |
       index | config | help | --help | --version}
```

The third is a general form as it explicitly exposes all commands (for virtual env creation, removal, discovery, etc.) Most of these commands have a shorter alias you'll probably prefer in everyday use (like `mkenv`, `lsenv`, `findenv`, `chenv`, `rmenv`, etc).

3.2 `envie` / `chenv` - Interactively activate the closest virtual environment

```
Interactively activate the closest Python virtual environment relative to DIR (or .)
A list of the closest environments is filtered by KEYWORDS. Separate KEYWORDS with --
if they start with a dash, or a dir with the same name exists.

Usage:
    chenv [-1] [-f|-l] [-v] [-q] [DIR] [--] [KEYWORDS]
    envie ...

Options:
    -1              activate only if a single closest env found, abort otherwise
    -f, --find      use only 'find' for search
    -l, --locate    use only 'locate' for search
    -v              be verbose: show info messages (path to activated env)
    -q              be quiet: suppress error messages
```

`chenv` command uses *findenv* to discover all virtual environments in `DIR`'s vicinity (searching below `DIR`, then dir-by-dir up until at least one virtual env is found), and then *fuzzy-filters* that list with a list of `KEYWORDS` given. If a single virtual environment is found, it's automatically activated. If multiple environments are found, user chooses the environment from a list.

3.2.1 Examples

Suppose you have a directory structure like this:

```
work
- plucky
|   - env
|   |   - dev
|   |   - prod
|   - src
- jsonplus
|   - pythonenv
|   - src
|   - var
```

Starting from base directory `work`, we can activate `jsonplus` environment with:

```
~/work$ envie js
Activated virtual environment at 'jsonplus/pythonenv'.
```

Or, starting from a project root at `work/jsonplus/src`, just type:

```
~/work/jsonplus/src$ envie
Activated virtual environment at '../pythonenv'.
```

When your query matches multiple environments, you'll get a prompt:

```
~/work$ envie plucky
1) plucky/env/dev
2) plucky/env/prod
#? 2
Activated virtual environment at 'plucky/env/prod'.
```

But you can avoid it by being a bit more specific:

```
~/work$ envie prrrod
Activated virtual environment at 'plucky/env/prod'.
```

(Notice we had a typo here, `prrrod`.)

3.3 `envie create` / `mkenv` - Create a new virtual environment

```
Create Python (2/3) virtual environment in DEST_DIR based on PYTHON.

Usage:
    mkenv [-2|-3|-e PYTHON] [-r PIP_REQ] [-p PIP_PKG] [-a] [-t] [DEST_DIR] [-- ARGS_
↪TO_VIRTUALENV]
    mkenv2 [-r PIP_REQ] [-p PIP_PKG] [-a] [-t] [DEST_DIR] ...
    mkenv3 [-r PIP_REQ] [-p PIP_REQ] [-a] [-t] [DEST_DIR] ...
    envie create ...

Options:
    -2, -3      use Python 2, or Python 3
    -e PYTHON   use Python accessible with PYTHON name,
                like 'python3.5', or '/usr/local/bin/mypython'.
    -r PIP_REQ  install pip requirements in the created virtualenv,
                e.g. '-r dev-requirements.txt'
    -p PIP_PKG  install pip package in the created virtualenv,
                e.g. '-p "Django>=1.9"', '-p /var/pip/pkg', '-p "-e git+https://gith..
↪."'
    -a          autodetect and install pip requirements
                (search for the closest 'requirements.txt' and install it)
    -t          create throw-away env in /tmp
    -v[v]       be verbose: show virtualenv&pip info/debug messages
    -q[q]       be quiet: suppress info/error messages
```

This command creates a new Python virtual environment (using the `virtualenv` tool) in the optionally supplied destination directory `DEST_DIR`. Default destination is `env` in the current directory, but that default can be overriden via *config variable* `_ENVIE_DEFAULT_ENVNAME`).

The default Python interpreter (executable used in a new virtual env) is defined with the config variable `_ENVIE_DEFAULT_PYTHON` and if not specified otherwise, it defaults to system `python`. Python executable can always be explicitly specified with `-e` parameter, e.g: `-e /path/to/python`, or `-e python3.5`. The shorthand flags `-2` and `-3` will select the default Python 2 and Python 3 interpreters available, respectively.

Tip: You can use aliases `mkenv2` and `mkenv3` instead of `mkenv -2` and `mkenv -3`, respectively.

To (pre-)install a set of Pip packages (requirements) in the virtual env created, you can use `-r` and `-p` options, like: `-r requirements.txt` and `-p package/archive/url`. The former will install requirements from a given file (or files, if option is repeated), and the latter will install a specific Pip package (or packages, if option repeated). The `-p` option supports all pip-supported formats: requirement specifier, VCS package URL, local package path, or archive path/URL:

- `-p requests`, `-p "jsonplus>=0.6"`,
- `-p /path/to/my/local/package`,
- `-p "-e git+https://github.com/randomir/plucky.git#egg=plucky"`.

If a standard name for requirements file is used in your project (`requirements.txt`), you can use the `-a` flag to find and auto-install the closest requirements below the CWD.

Throw-away or temporary environment is created with `-t` flag. The location and name of the virtual environment are chosen randomly with the `mktemp` (something like `/tmp/tmp.4Be8JJ8OJb`). When done with hacking in a throw-away env, simply destroy it with `rmenv -f`.

Tip: Throw-away environments are great for short-lived experiments, for example:

```
$ mkenv3 -t -p requests -p plucky && python && rmenv -fv
Creating Python virtual environment in '/tmp/tmp.ial0H5kZvu'.
Using Python 3.5.2+ (/usr/bin/python3).
Virtual environment ready.
Installing Pip requirements: requests plucky
Pip requirements installed.
Python 3.5.2+ (default, Sep 22 2016, 12:18:14)
[GCC 6.2.0 20160927] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests, plucky
>>> plucky.pluck(requests.get('https://api.github.com/users/randomir/repos').json(),
↪'name')
['blobber', 'dendritic-growth-model', 'envie', 'joe', 'jsonplus', 'python-digitalocean
↪', ...]
>>> exit()
VirtualEnv removed: /tmp/tmp.ial0H5kZvu
```

3.3.1 Examples

Starting from a base directory `~/work`, let's create Python 2 & 3 virtual environments for our new project `yakkety`:

```
~/work$ mkenv3 yakkety/env/dev
Creating Python virtual environment in 'yakkety/env/dev'.
Using Python 3.5.2+ (/usr/bin/python3).
Virtual environment ready.
(dev) ~/work$

(dev) ~/work$ mkenv2 yakkety/env/dev
Creating Python virtual environment in 'yakkety/env/prod'.
Using Python 2.7.12+ (/usr/bin/python2).
Virtual environment ready.
(prod) ~/work$
```

Note here (1) directory structure is recursively created, and (2) active environment does not interfere with Python interpreter discovery.

We can create a temporary environment with dev version of package installed from GitHub source:

```
$ mkenv -tp "-e git+https://github.com/randomir/plucky.git#egg=plucky"
```

3.4 `envie remove` / `rmenv` - Delete the active virtual environment

```
Remove (delete) the base directory of the active virtual environment.

Usage:
    rmenv [-f] [-v]
    envie remove ...

Options:
    -f    force; don't ask for permission
    -v    be verbose
```

`rmenv` will remove a complete virtual env directory tree of the active environment (defined with shell variable `$VIRTUAL_ENV`), or fail otherwise. To avoid prompting for confirmation, supply the `-f` flag, and to print the directory removed, use the `-v` switch.

3.5 `envie list` / `lsenv [DIR]` - List virtual environments below `DIR`

```
Find and list all virtualenvs under DIR, optionally filtered by KEYWORDS.

Usage:
    lsenv [-f|-l] [DIR [AVOID_SUBDIR]] [--] [KEYWORDS]
    envie list ...

Options:
    -f, --find    use only 'find' for search
    -l, --locate  use only 'locate' for search
                  (by default, try find for 0.4s, then failback to locate)
    -v            be verbose: show info messages
    -q            be quiet: suppress error messages
```

`envie list` searches down only, starting in `DIR` (defaults to `.`). The search method is defined with config, but it can be overriden with `-f` and `-l` to force `find` or `locate` methods respectively. **Fuzzy filtering.** To narrow down the list of virtualenv paths, you can filter it by supplying `KEYWORDS`. Filtering algorithm is not strict and exclusive (like grep), but fuzzy and typo-forgiving.

It works like this: (1) all virtualenv paths discovered are split into directory components; (2) we try to greedily match all keywords to components by maximum similarity score; (3) paths are sorted by total similarity score; (4) the best matches are passed-thru – if there's a tie, all best matches are printed.

When calculating similarity between directory name (path component) and a keyword, we assign: (1) maximum weight to a complete match (identity), (2) smaller, but still high, weight to a prefix match, and (3) the smallest (and variable) weight to a diff-metric similarity.

3.5.1 Examples

For an example, suppose you have a directory tree like this one:

```
- trusty-tahr
|   - dev
|   - prod
- zesty-zapus
|   - dev
|   - prod
```

To get all environments containing `dev` word:

```
$ lsenv dev
trusty-tahr/dev
zesty-zapus/dev
```

To get all `trusty` envs, you can either filter by `trusty` (or `tahr`, or `hr`, or `t`):

```
$ lsenv hr
trusty-tahr/dev
trusty-tahr/prod
```

or, list envs in `./trusty-tahr` dir:

```
$ lsenv ./trusty-tahr
trusty-tahr/dev
trusty-tahr/prod
```

Combine it:

```
$ lsenv trusty-tahr pr
trusty-tahr/prod
```

or with several keywords:

```
$ lsenv z d
zesty-zapus/dev
```

3.6 `envie find` / `findenv [DIR]` - Find the closest virtual environment around `DIR`

```
Find and list all virtualenvs below DIR, or above if none found below.
List of virtualenv paths returned is optionally filtered by KEYWORDS.

Usage:
    findenv [-f|-l] [DIR] [--] [KEYWORDS]
    envie find ...

Options:
    -f, --find    use only 'find' for search
    -l, --locate  use only 'locate' for search
                  (by default, try find for 0.4s, then failback to locate)
    -v            be verbose: show info messages
    -q            be quiet: suppress error messages
```

Similar to `envie list`, but with a key distinction: if no environments are found below the starting `DIR`, the search is being expanded – level by level up – until at least one virtual environment is found.

Description of discovery methods (`--find`/`--locate`), as well as keywords filtering behaviour given for `envie list`/`lsenv` apply here also.

3.7 `envie python` / `envie SCRIPT` - Run Python SCRIPT in the closest virtual environment

Run a Python SCRIPT, or an interactive Python interpreter session in the closest virtual environment. Three calling forms are supported:

`envie SCRIPT [ARGS]` The `SCRIPT` is explicitly executed with `python` from the closest environment. If multiple environments are found in the vicinity, operation is aborted.

`envie python SCRIPT [ARGS]` Identical in behaviour to the above, but more explict.

`envie python` A special no-script case, where an interactive Python session is started instead.

Hint: This command is basically a shortcut for:

```
chenv -1v && exec python [SCRIPT [ARGS]]
```

3.7.1 Examples

```
envie manage.py migrate

envie python tests.py --fast
```

3.8 `envie run CMD` - Run CMD in the closest virtual env

As a generalization of the `envie python` command above, this command will run *anything that can be run*, in the closest virtual environment. The `CMD` can be an **executable** (script) file, shell **command**, shell **builtin**, **alias**, or a shell **function**.

`envie run CMD [ARGS]` Runs any executable construct `CMD`, optionally passing-thru arguments `ARGS`, inside the closest virtual environment. Fails when multiple environments are found in the vicinity.

Hint: Similarly to `envie python`, this command is basically a shortcut for:

```
chenv -1v && exec CMD [ARGS]
```

3.8.1 Examples

We can emulate the `envie python` command with:

```
envie run python /path/to/my/script
```

but also run shell functions which are sensitive to the Python virtual env:

```
envie run my_function
```

Moreover, we can run the apropriate `python` in the command mode:

```
envie run python -c 'import os; print(os.getenv("VIRTUAL_ENV"))'
```

3.9 `envie config` - Configure Envie

Listed here for completeness, configuration is described in detail under the *Configure* section.

3.10 `envie index` - (Re-)Index Environments

If Envie is configured to use `locate` for environments discovery, index can be rebuilt (updated via `updatedb`) with `envie index`. For more on `find` vs. `locate` methods, see *here*.

3.11 `envie-tmp SCRIPT` - Run SCRIPT in a temporary environment

```
Create a new temporary (throw-away) virtual environment, install requirements
specified, run the SCRIPT, and destroy the environment afterwards.

Usage:
    envie-tmp SCRIPT

Hashbang examples:

 1) no requirements (mkenv -t)

    #!/usr/bin/env envie-tmp

 2) installs reqs from the closest "requirements.txt" (mkenv -ta):

    #!/usr/bin/env envie-tmp
    # -*- requirements: auto -*-

 3) installs reqs from the specific Pip requirements files (relative to SCRIPT's dir)
    (mkenv -t -r REQ ...):

    #!/usr/bin/env envie-tmp
    # -*- requirements: ../base-requirements.txt ./dev-requirements.txt -*-

 4) specify the Python version to use, and install some Pip packages
    (mkenv -t -e PYTHON -p PKG ...):

    #!/usr/bin/env envie-tmp
    # -*- python-version: python3 -*-
    # -*- packages: plucky requests>=2.0 flask==0.12 -e/path/to/pkg -e. -*-
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search