Dram Documentation

Release 0.0.3.dev

Scott Torborg

Aug 06, 2021

Contents

Author: Scott Torborg

Dram is a POSIX-specific tool for managing shell environments. The basic goal is to make it possible to install new shell-based software, from a number of different sources, with the confidence that it won't mess up your global environment.

Instead of simply installing new software into `/usr/local`, you can quickly activate a new "dram", and install it there. If something goes wrong, no problem: just delete the entire dram.

You can also create new drams that contain an instance of Homebrew or Macports. This makes it easy to use both Homebrew and Macports at the same time, or multiple instances of either. Again, if somsething goes wrong, nothing to worry about: just delete the dram.

CHAPTER 1

Contents

1.1 Basics

1.1.1 Concepts

Each new sandboxed environment is referred to as a *dram*.

The environment variable `DRAM_ROOT` is the root directory where all drams are kept. Each dram is a subdirectory.

> **Warning:** Root privileges should generally not be necessary to install anything into a dram. Avoid the temptation to use `sudo`. You may wish to consider changing the ownership of your `/usr/local` directory to root in order to prevent any dram usage from "spilling" into the global environment.

1.1.2 Dram Types

- **Plain** drams are intended for installing most open source "directly", via a `./configure` script or similar mechanism. They are not populated with any initial files or directories, but set up environment variables that will
- **Plain-with-python** drams are the same as Plain drams with the addition of an integrated python virtual environment for the dram.
- **Homebrew** drams populate an instance of Homebrew.
- **Macports** drams populate an instance of MacPorts.

1.1.3 Command-Line Usage

Dram includes a command-line utility which is the expected entry point for most usage. Some examples are below. You can also see the utility itself for more info:

```
$ dram help
```

1.1.4 Creating a New Dram

This will create a new dram you can then install stuff into.:

```
$ dram create example
```

To create a dram of a particular type:

```
$ dram create -t macports example
```

When creating a dram of type `plain-with-python` you can also pass the `-p` and `--system-site-packages` flags and they will be passed through to the virtual environement when it is created for that dram.

1.1.5 Using The Dram

Creating a dram automatically switches to it, but if you want to switch to an already created dram:

```
$ dram use example
```

Note that if you have set the `DRAM_AUTO_CDSOURCE` environment variable, then dram will automatically switch to the source directory of the activated dram whenever you do `dram use <dram name>`

Dram includes wrappers around common build tools (autotools and cmake) to ease installing software into the dram.

1.1.6 Installing Software with CMake

As an example, let's install *libftdi* from a source tarball.

First move into the *source* subdirectory, by convention:

```
$ dram cdsource
```

Download and extract the package here:

```
$ curl -O https://www.intra2net.com/en/developer/libftdi/download/libftdi1-1.5.tar.bz2
$ tar -xvf libftdi1-1.5.tar.bz2
$ cd libftdi1-1.5
```

Invoke CMake using the dram wrapper:

```
$ dram cmake ..
```

Install normally, which will install into this dram's directory structure:

```
$ make && make install
```

1.1.7 Install Software with Autotools

Install autotools-based software is similar. Here is a typical installation of *liquid-sdr*:

```
$ dram cdsource
$ git clone https://github.com/jgaeddert/liquid-dsp.git
$ cd liquid-dsp
$ ./bootstrap.sh
$ dram configure
$ make && make install
```

1.1.8 Listing Drams

List all the current drams in your root:

```
$ dram list
```

If the `-l` flag is passed to `dram list`, then the size of each dram on disk will be printed as well.

1.1.9 Destroying a Dram

Wipe out any traces of a current dram:

```
$ dram destroy example
```

1.2 Extending Behavior with Hooks

You can extend dram to add special user-specific behavior with specially-named function hooks. To implement a hook, simply define a function with the hook's name in your shell environment (for example, in `.bashrc`).

1.2.1 Example

Set the shell prompt to include the name of the activated dram, upon activation:

```
function dram_hook_postactivate() {
    local dram_name=$1
    local dram_prefix=$2

    PS1="($local_dram_name) $PS1"
}
```

Deactivate a Python `virtualenv` before activating a dram, if one is currently active:

```
function deactivate_any_virtualenv () {
    type deactivate >/dev/null 2>&1
    if [ $? -eq 0 ]
    then
        deactivate
    fi
}

function dram_hook_preactivate () {
    local dram_name=$1
    local dram_prefix=$2
```

(continues on next page)

(continued from previous page)

```
    deactivate_any_virtualenv
}
```

1.2.2 Available Hooks

Currently available hooks are:

`dram_hook_preactivate`: Called prior to activating a dram, with the name and path prefix of the dram.

`dram_hook_postactivate`: Called immediately after activating a drone, with the name and path prefix of the dram.

1.3 Contributing

Patches and suggestions are strongly encouraged! GitHub pull requests are preferred, but other mechanisms of feedback are welcome.

CHAPTER 2

Indices and Tables

- genindex
- modindex