dwim

Release 0.3.1

May 29, 2017

Contents

The `dwim` program is a location aware application launcher. To use it you are required to create a profile at `~/.dwimrc`. This profile is a simple Python script that defines which applications you want to start automatically, in which order the applications should start and whether some applications should only be started when your computer is on a specific physical location. The location awareness works by matching a unique property of the network that your computer is connected to (the MAC address of your current network gateway).

Every time you run the `dwim` program your `~/.dwimrc` profile is evaluated and your applications are started automatically. If you run `dwim` again it will not start duplicate instances of your applications, but when you quit an application and then rerun `dwim` the application will be started again.

CHAPTER 1

Installation

The *dwim* package is available on PyPI which means installation should be as simple as:

```
$ pip install dwim
```

There's actually a multitude of ways to install Python packages (e.g. the per user site-packages directory, virtual environments or just installing system wide) and I have no intention of getting into that discussion here, so if this intimidates you then read up on your options before returning to these instructions ;-).

CHAPTER 2

Usage

There are two ways to use the *dwim* package: As the command line program `dwim` and as a Python API. For details about the Python API please refer to the API documentation available on Read the Docs. The command line interface is described below.

Please note that you need to *create a profile* (see below) before you can use the program.

Command line interface

Usage: *dwim [OPTIONS]*

The dwim program is a location aware application launcher. To use it you are required to create a profile at ~/.dwimrc. This profile is a simple Python script that defines which applications you want to start automatically, in which order the applications should start and whether some applications should only be started given a specific physical location.

The location awareness works by checking the MAC address of your gateway (the device on your network that connects you to the outside world, usually a router) to a set of known MAC addresses that you define in ~/.dwimrc.

Every time you run the dwim program your ~/.dwimrc profile is evaluated and your applications are started automatically. If you run dwim again it will not start duplicate instances of your applications, but when you quit an application and then rerun dwim the application will be started again.

Supported options:

Option	Description
`-c, --config=FILE`	Override the default location of the profile script.
`-v, --verbose`	Increase logging verbosity (can be repeated).
`-q, --quiet`	Decrease logging verbosity (can be repeated).
`-h, --help`	Show this message and exit.

Creating a profile

To use `dwim` you need to create a profile at `~/.dwimrc`. The profile is a simple Python script that defines which applications you want to start automatically, in which order the applications should start and whether some applications should only be started on a specific physical location. The profile script has access to functions provided by the `dwim` Python package. Please refer to the documentation for the available functions. The examples below show the most useful functions.

Starting your first program

If you'd like to get your feet wet with a simple example, try this:

```
launch_program('pidgin')
```

When you've created the above profile script and you run the `dwim` program it will start the Pidgin chat client on the first run. On the next run nothing will happen because Pidgin is already running.

Modifying the "is running" check

The default "is running" check comes down to the following shell command line:

```
# Replace `pidgin' with any program name.
pidof $(which pidgin)
```

This logic will not work for all programs. For example in my profile I start the Dropbox client using a wrapper script. Once the Dropbox client has been started the wrapper script terminates so the `pidof` check fails. The solution is to customize the "is running" check:

```
launch_program('dropbox start', is_running='pgrep -f "$HOME/.dropbox-dist/*/dropbox"')
```

The example above is for the Dropbox client, but the same principle can be applied to all other programs. The only trick is to find a shell command that can correctly tell whether the program is running. Unfortunately this part cannot be automated in a completely generic manner. The advanced profile example below contains more examples of defining custom `pidof` checks and `pgrep -f` checks.

Enabling location awareness

The first step to enabling location awareness is to add the following line to your profile:

```
determine_network_location()
```

Even if you don't pass any information to this function it will still report your current gateway's MAC address. This saves me from having to document the shell commands needed to do the same thing :-). Run the `dwim` command and take note of a line that looks like this:

```
We're not connected to a known network (unknown gateway MAC address␣
↪84:9c:a6:76:23:8e).
```

Now edit your profile and change the line you just added:

```
location = determine_network_location(home=['84:9c:a6:76:23:8e'])
```

When you now rerun `dwim` it will say:

```
We're connected to the home network.
```

So what did we just do? We took note of the current gateway's MAC address and associated that MAC address with a location named "home". In our profile we can now start programs on the condition that we're connected to the home network:

```
if location == 'home':
   # Client for Music Player Daemon.
   launch_program('ario --minimized')
else:
   # Standalone music player.
   launch_program('rhythmbox')
```

The example profile below (my profile) contains a more advanced example combining multiple networks and networks with multiple gateways.

Example profile

I've been using variants of `dwim` (previously in the form of a Bash script :-) for years now so my profile has grown quite a bit. Because of this it may provide some interesting examples of things you can do:

```
# vim: fileencoding=utf-8

# ~/.dwimrc: Profile for dwim, my location aware application launcher.
# For more information please see https://github.com/xolox/python-dwim/.

# Standard library modules.
import os
import time

# Packages provided by dwim and its dependencies.
from executor import execute
from dwim import (determine_network_location, launch_program, LaunchStatus
                  set_random_background, wait_for_internet_connection)

# This is required for graphical Vim and gnome-terminal to have nicely
# anti-aliased fonts. See http://awesome.naquadah.org/wiki/Autostart.
if launch_program('gnome-settings-daemon') == LaunchStatus.started:

    # When my window manager is initially started I need to wait for a moment
    # before launching user programs because otherwise strange things can
    # happen, for example programs that place an icon in the notification area
    # might be started in the background without adding the icon, so there's
    # no way to access the program but `dwim' will never restart the program
    # because it's already running! _
    logger.debug("Sleeping for 10 seconds to give Awesome a moment to initialize ..")
    time.sleep(10)
```

```
# Determine the physical location of this computer by matching the MAC address
# of the gateway against a set of known MAC addresses. In my own copy I've
# documented which MAC addresses belong to which devices, but that doesn't seem
# very relevant for the outside world :-)
location = determine_network_location(home=['84:9C:A6:76:23:8E'],
                                      office=['00:15:C5:5F:92:79',
                                              'B6:25:B2:19:28:61',
                                              '00:18:8B:F8:AF:33'])

# Correctly configure my multi-monitor setup based on physical location.
if location == 'home':
    # At home I use a 24" ASUS monitor as my primary screen.
    # My MacBook Air sits to the left as the secondary screen.
    execute('xrandr --output eDP1 --auto --noprimary')
    execute('xrandr --output HDMI1 --auto --primary')
    execute('xrandr --output HDMI1 --right-of eDP1')
if location == 'work':
    # At work I use a 24" LG monitor as my primary screen.
    # My Asus Zenbook sits to the right as the secondary screen.
    execute('xrandr --output eDP1 --auto')
    execute('xrandr --output HDMI1 --auto')
    execute('xrandr --output HDMI1 --left-of eDP1')

# Set a random desktop background from my collection of wallpapers. I use the
# program `feh' for this because it supports my desktop environment / window
# manager (Awesome). You can install `feh' using `sudo apt-get install feh'.
set_random_background(command='feh --bg-scale {image}',
                      directory=os.path.expanduser('~/Pictures/Backgrounds'))

# Start my favorite programs.
launch_program('gvim')
launch_program('nm-applet')
launch_program('keepassx $HOME/Documents/Passwords/Personal.kdb -min -lock',
               is_running='pgrep -f "keepassx $HOME/Documents/Passwords/Personal.kdb"
↪')
# I actually use three encrypted key passes, two of them for work. I omitted
# those here, but their existence explains the complex is_running command.
launch_program('fluxgui', is_running='pgrep -f $(which fluxgui)')

# The remaining programs require an active internet connection.
wait_for_internet_connection()

launch_program('chromium-browser', is_running='pidof /usr/lib/chromium-browser/
↪chromium-browser')
launch_program('pidgin')
if location == 'home':
    # Mozilla Thunderbird is only useful at home (at work IMAPS port 993 is blocked).
    launch_program('thunderbird', is_running='pidof /usr/lib/thunderbird/thunderbird')
launch_program('dropbox start', is_running='pgrep -f "$HOME/.dropbox-dist/*/dropbox"')
launch_program('spotify')
```

CHAPTER 3

Location awareness

The location awareness works by matching the MAC address of your current network gateway (your router). I've previously also used public IPv4 addresses but given the fact that most consumers will have a dynamic IP address I believe the gateway MAC access is the most stable unique property to match.

CHAPTER 4

About the name

In programming culture the abbreviation DWIM stands for Do What I Mean. The linked Wikipedia article refers to Interlisp but I actually know the term from the world of Perl. The reason I chose this name for my application launcher is because I like to make computer systems anticipate what I want. Plugging in a network cable, booting my laptop and having all my commonly used programs (depending on my physical location) instantly available at startup is a great example of Do What I Mean if you ask me :-)

CHAPTER 5

Contact

The latest version of *dwim* is available on PyPI and GitHub. The documentation is hosted on Read the Docs. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

CHAPTER 6

License

CHAPTER 7

Function reference

The following documentation is based on the source code of version 0.3.1 of the `dwim` package.

dwim

dwim: Location aware application launcher.

`dwim.`**`DEFAULT_PROFILE = '~/.dwimrc'`**
The default location of the user's profile script (a string).

`dwim.`**`dwim`**(*profile='~/.dwimrc'*)
Evaluate the user's profile script.

`dwim.`**`launch_program`**(*command*, *is_running=None*)
Start a program if it's not already running.

This function makes it easy to turn any program into a single instance program. If the default "Is the program already running?" check fails to work you can redefine the way this check is done.

Parameters

- **`command`** – The shell command used to launch the application (a string).
- **`is_running`** – The shell command used to check whether the application is already running (a string, optional).

Returns One of the values from the *`LaunchStatus`* enumeration.

Examples of custom "is running" checks:

```
# Chromium uses a wrapper script, so we need to match the absolute
# pathname of the executable.
launch_program('chromium-browser', is_running='pidof /usr/lib/chromium-browser/
↪chromium-browser')

# Dropbox does the same thing as Chromium, but the absolute pathname of
# the executable contains a version number that I don't want to hard
```

```
# code in my ~/.dwimrc profile :-)
launch_program('dropbox start', is_running='pgrep -f "$HOME/.dropbox-dist/*/
↪dropbox"')
```

class dwim.**LaunchStatus**

LaunchStatus enumerates the possible results of *launch_program()*.

It enables the caller to handle the possible results when they choose to do so, without forcing them to handle exceptions.

started = <EnumValue: LaunchStatus.started [value=1]>
The program wasn't running before but has just been started.

already_running = <EnumValue: LaunchStatus.already_running [value=2]>
The program was already running.

not_installed = <EnumValue: LaunchStatus.not_installed [value=3]>
The program is not installed / available on the $PATH.

unspecified_error = <EnumValue: LaunchStatus.unspecified_error [value=4]>
Any other type of error, e.g. the command line can't be parsed.

dwim.**extract_program**(*command_line*)

Parse a simple shell command to extract the program name.

Parameters **command_line** – A shell command (a string).

Returns The program name (a string).

Raises *CommandParseError* when the command line cannot be parsed.

Some examples:

```
>>> extract_program('dropbox start')
'dropbox'
>>> extract_program(' "/usr/bin/dropbox" start ')
'/usr/bin/dropbox'
```

dwim.**resolve_program**(*executable*)

Expand the name of a program into an absolute pathname.

Parameters **executable** – The name of a program (a string).

Returns The absolute pathname of the program (a string).

Raises *MissingProgramError* when the program doesn't exist.

An example:

```
>>> extract_program('dropbox start')
'dropbox'
>>> resolve_program(extract_program('dropbox start'))
'/usr/bin/dropbox'
```

dwim.**set_random_background**(*command*, *directory*)

Set a random desktop wallpaper / background.

Parameters

- **command** – The command to set the wallpaper (a string containing an {image} marker).
- **directory** – The pathname of a directory containing wallpapers (a string).

Raises ValueError when the *command* string doesn't contain an {image} placeholder.

dwim.**determine_network_location**(***gateways*)
Determine the physical location of this computer.

This works by matching the MAC address of the current gateway against a set of known MAC addresses, which provides a simple but robust way to identify the current network. Because networks usually have a physical location, identifying the current network tells us our physical location.

Parameters **gateways** – One or more keyword arguments with lists of strings containing MAC addresses of known networks.

Returns The name of the matched MAC address (a string) or `None` when the MAC address of the current gateway is unknown.

Here's an example from my `~/.dwimrc` involving multiple networks and a physical location with multiple gateways:

```
location = determine_network_location(home=['84:9C:A6:76:23:8E'],
                                      office=['00:15:C5:5F:92:79',
                                              'B6:25:B2:19:28:61',
                                              '00:18:8B:F8:AF:33'])
```

dwim.**find_gateway_address**()
Find the IP address of the current gateway using the `ip route` command.

Returns The IP address of the gateway (a string) or `None`.

An example:

```
>>> find_gateway_address()
'192.168.1.1'
```

dwim.**find_gateway_mac**()
Find the MAC address of the current gateway using the `arp -n` command.

Returns The MAC address of the gateway (a string) or `None`.

An example:

```
>>> find_gateway_address()
'192.168.1.1'
>>> find_gateway_mac(find_gateway_address())
'84:9c:a6:76:23:8e'
```

dwim.**wait_for_internet_connection**()
Wait for an active internet connection.

This works by sending `ping` requests to `8.8.8.8` (one of the Google public DNS IPv4 addresses) and returning as soon as a ping request gets a successful response. The ping interval and timeout is one second.

dwim.**have_internet_connection**()
Check if an internet connection is available.

Returns `True` if an internet connection is available, `False` otherwise.

This works by pinging 8.8.8.8 which is one of Google's public DNS servers. This IP address was chosen because it is documented that Google uses anycast to keep this IP address available at all times.

dwim.cli

Usage: *dwim [OPTIONS]*

The dwim program is a location aware application launcher. To use it you are required to create a profile at ~/.dwimrc. This profile is a simple Python script that defines which applications you want to start automatically, in which order the applications should start and whether some applications should only be started given a specific physical location.

The location awareness works by checking the MAC address of your gateway (the device on your network that connects you to the outside world, usually a router) to a set of known MAC addresses that you define in ~/.dwimrc.

Every time you run the dwim program your ~/.dwimrc profile is evaluated and your applications are started automatically. If you run dwim again it will not start duplicate instances of your applications, but when you quit an application and then rerun dwim the application will be started again.

Supported options:

Option	Description
`-c`, `--config=FILE`	Override the default location of the profile script.
`-v`, `--verbose`	Increase logging verbosity (can be repeated).
`-q`, `--quiet`	Decrease logging verbosity (can be repeated).
`-h`, `--help`	Show this message and exit.

`dwim.cli.`**`main`**`()`
Command line interface for the `dwim` program.

dwim.exceptions

Custom exceptions raised by *dwim*.

exception `dwim.exceptions.`**`ProgramError`**
Super class for exceptions raised in *launch_program()*.

exception `dwim.exceptions.`**`CommandParseError`**
Raised by *extract_program()* when a command line can't be parsed or is empty.

exception `dwim.exceptions.`**`MissingProgramError`**
Raised by *resolve_program()* when a program doesn't exist.

Python Module Index

d

Index