wifi Documentation

Release 0.2.0

Rocky Meza and Gavin Wahl

Mar 20, 2017

Contents

Note: This project is unmaintained. For more information, please visit the GitHub page.

Wifi provides a set of tools for configuring and connecting to WiFi networks on Linux systems. Using this library, you can discover networks, connect to them, save your configurations, and much, much more.

The original impetus for creating this library was my frustration with with connecting to the Internet using NetworkManager and wicd. It is very much for computer programmers, not so much for normal computer users. Wifi is built on top the old technologies of the */etc/network/interfaces* file and *ifup* and *ifdown*. It is inspired by *ifscheme*.

The library also comes with an executable that you can use to manage your WiFi connections. Wifi currently supports the following encryption types:

- No encryption
- WEP
- WPA2

If you need support for other network types, please file a bug on GitHub and we'll definitely try to get to it. Patches, of course, are always welcome.

CHAPTER 1

Installation

Wifi is available for installation on PyPI:

```
$ pip install wifi
```

This will install the *the wifi command*, a Python library for discovering and connecting to wifi networks, and a bash completion file for the wifi command.

On some systems, the wifi command name is already used, and installing wifi will cause issues with the system. In those cases you can override the command name that is installed:

```
$ WIFI_CLI_NAME=cool-wifi pip install wifi
```

The wifi executable <*wifi_command*> will instead be named *cool-wifi*.

If you only want the Python library and don't care about the CLI, you can also disable it:

```
$ WIFI_INSTALL_CLI=False pip install wifi
```

There will be no extra executable installed, but the wifi CLI will still be available using this invocation style:

```
$ python -m wifi
```

CHAPTER 2

Documentation

The wifi Command

This library comes with a command line program for managing and saving your WiFi connections.

Tutorial

This tutorial assumes you are comfortable on the command line. (If you aren't, perhaps wifi is not quite the right tool for you.)

First, if you haven't already, install wifi.

```
$ pip install wifi
```

Now, you want to see what networks are available. You can run the `scan` command to do that.

Note: All of these commands need to run as a superuser.

```
# wifi scan
-61  SomeNet                  protected
-62  SomeOtherNet             unprotected
-78  zxy-12345                protected
-86  TP-LINK_CB1676           protected
-86  TP-LINK_PocketAP_D8B616  unprotected
-82  TP-LINK_C1DBE8           protected
-86  XXYYYYZZZ                protected
-87  Made Up Name             protected
```

Note: The wifi command is also accessible through the python argument as:

```
# python -m wifi
```

The scan command returns three bits of data: the signal quality, the SSID and if the network is protected or not. If you want to order the networks by quality, you can pipe the output into sort.

```
# wifi scan | sort -rn
-61  SomeNet                  protected
-62  SomeOtherNet             unprotected
-78  zxy-12345                protected
-82  TP-LINK_C1DBE8           protected
-86  XXYYYYZZZ                protected
-86  TP-LINK_PocketAP_D8B616  unprotected
-86  TP-LINK_CB1676           protected
-87  Made Up Name             protected
```

The greater the number, the better the signal.

We decide to use the `SomeNet` network because that's the closest one (plus we know the password). We can connect to it directly using the `connect` command.

```
# wifi connect --ad-hoc SomeNet
passkey>
```

The `--ad-hoc` or `-a` option allows us to connect to a network that we haven't configured before. The wifi asks you for a passkey if the network is protected and then it will connect.

If you want to actually save the configuration instead of just connecting once, you can use the `add` command.

```
# wifi add some SomeNet
passkey>
```

`some` here is a nickname for the network you can use when you want to connect to the network again. Now we can connect to the saved network if you want using the `connect` command.

```
# wifi connect some
...
```

If you wish to see all the saved networks, you can use the `list` command.

```
# wifi list
some
```

Usage

```
usage: wifi {scan,list,config,add,connect,init} ...
```

scan

Shows a list of available networks.

```
usage: wifi scan
```

list

Shows a list of networks already configured.

```
usage: wifi list
```

add, config

Prints or adds the configuration to connect to a new network.

```
usage: wifi config SCHEME [SSID]
usage: wifi add SCHEME [SSID]

positional arguments:
  SCHEME      A memorable nickname for a wireless network. If SSID is not
              provided, the network will be guessed using SCHEME.
  SSID        The SSID for the network to which you wish to connect. This is
              fuzzy matched, so you don't have to be precise.
```

connect

Connects to the network corresponding to SCHEME.

```
usage: wifi connect [-a] SCHEME

positional arguments:
  SCHEME        The nickname of the network to which you wish to connect.

optional arguments:
  -a, --ad-hoc  Connect to a network without storing it in the config file
```

autoconnect

Searches for saved schemes that are currently available and connects to the first one it finds.

```
usage: wifi autoconnect
```

Completion

The wifi command also comes packaged with completion for bash. If you want to write completion for your own shell, wifi provides an interface for extracting completion information. Please see the `wifi-completion.bash` and `bin/wifi` files for more information.

Managing WiFi networks

Discovering networks

You can use this library to scan for networks that are available in the air. To get a list of the different cells in the area, you can do

```
>>> from wifi import Cell, Scheme
>>> Cell.all('wlan0')
```

This returns a list of *Cell* objects. Under the hood, this calls *iwlist scan* and parses the unfriendly output.

Each cell object should have the following attributes:

- ssid
- signal
- quality
- frequency
- bitrates
- encrypted
- channel
- address
- mode

For cells that have encrypted as *True*, there will also be the following attributes:

- encryption_type

Note: Scanning requires root permission to see all the networks. If you are not root, iwlist only returns the network you are currently connected to.

Connecting to a network

In order to connect to a network, you need to set up a scheme for it.

```
>>> cell = Cell.all('wlan0')[0]
>>> scheme = Scheme.for_cell('wlan0', 'home', cell, passkey)
>>> scheme.save()
>>> scheme.activate()
```

Once you have a scheme saved, you can retrieve it using *Scheme.find()*.

```
>>> scheme = Scheme.find('wlan0', 'home')
>>> scheme.activate()
```

Note: Activating a scheme will disconnect from any other scheme before connecting.

You must be root to connect to a network. Wifi uses *ifdown* and *ifup* to connect and disconnect.

---

class wifi.**Cell**

Presents a Python interface to the output of iwlist.

classmethod all(*interface*)

Returns a list of all cells extracted from the output of iwlist.

classmethod from_string(*cell_string*)

Parses the output of iwlist scan for one cell and returns a Cell object for it.

classmethod **where**(*interface*, *fn*)
Runs a filter over the output of *all()* and the returns a list of cells that match that filter.

class wifi.**Scheme**(*interface*, *name*, *options=None*)
Saved configuration for connecting to a wireless network. This class provides a Python interface to the /etc/network/interfaces file.

activate()
Connects to the network as configured in this scheme.

classmethod **all**()
Returns an generator of saved schemes.

delete()
Deletes the configuration from the interfaces file.

classmethod **find**(*interface*, *name*)
Returns a *Scheme* or *None* based on interface and name.

classmethod **for_cell**(*interface*, *name*, *cell*, *passkey=None*)
Intuits the configuration needed for a specific *Cell* and creates a *Scheme* for it.

classmethod **for_file**(*interfaces*)
A class factory for providing a nice way to specify the interfaces file that you want to use. Use this instead of directly overwriting the interfaces Class attribute if you care about thread safety.

save()
Writes the configuration to the interfaces file.

Changelog

0.4.0

release-date unknown

- Make the wifi command name configurable (#55 - thanks yourealwaysbe)
- Add a __main__.py so that wifi can be invoked using python -mwifi
- Fix argument parsing so that scan is the default argument even with options passed

0.3.8

release-date 2016-03-11

- Parse noise level if available (#91 - thanks zgoda-mobica)

0.3.7

release-date 2016-03-11

- Fix bugs related to scheme parsing (#59, #42)

0.3.6

release-date 2016-02-11

- Set all attributes to None in Cell.__init__ (#88 - thanks stvad)

0.3.5

release-date 2016-01-24

- Better password handling (#62 - thanks foosel)
- Account for Cells with no SSID (#86 - thanks tlau)

0.3.4

release-date 2014-09-02

- Fixed installation missing some files (#48 - thanks luckydonald)

0.3.3

release-date 2014-08-31

- Check for write access for bashcompletion via os.access (#41, #47 - thanks foosel and jegger)
- Fixed scanning when quality is reported absolutely (#45 - jeromelebel)
- Fixed channel parsing (#33, #39 - thanks gavinwahl and qizha)

0.3.2

release-date 2014-07-26

- Only run if __name__ == '__main__' (#29 - thanks Jonwei)
- Try to connect to the nearest Access Point
- wifi scan was failing when Bit Rate was the last line of output (#42 - thanks jargij)
- Added documentation for signal and quality on Cell

0.3.1

release-date 2014-02-10

- Scheme.activate was failing on a TypeError in Python3

0.3.0

release-date 2014-02-09

- Scheme.activate now throws a ConnectionError if activation failed (#17 - thanks alexykot)
- Cell.all now throws an InterfaceError if scanning failed (#18 - thanks alexykot)

- Better error message when scheme isn't found (#19 - thanks gavinwahl)
- Added ability to delete schemes (#20 - thanks spektom)
- Added the –file option (#21)
- Scheme.activate returns a Connection object (#22)
- Added the autoconnect command (#23)
- Fixed parsing error missing channel (#24 - thanks LiorKirsch)
- Fixed relative signal return as zero (#25 - thanks LiorKirsch)
- Relative signals are now converted to dBm (#26 - thanks LiorKirsch)
- Various codebase cleanup (#27 - thanks ramnes)
- Added support for WPA Version 1 (#28 - thanks LiorKirsch)
- Fixed Python3 support for WPA/PBKDF2

0.2.2

release-date 2013-12-25

- Fixed relative signal parsing bug (#12 - thanks alexykot)

0.2.1

release-date 2013-11-22

- Fixed print_table str/int bug (#13 - thanks DanLipsitt)

0.2.0

release-date 2013-09-27

- Added support for WEP
- Fixed bug related to very short SSIDs
- Fixed bug related to numeric passkeys

0.1.1

release-date 2013-05-26

- Updated setup.py to actually install the bash completion script

CHAPTER 3

Contributing

The (very little) development for wifi happens on GitHub. If you ever run into issues with wifi, please don't hesitate to open an issue. Pull requests are welcome.

Index