easy_install: The Old-Fashioned But Easy Way to Install Python Packages

Andrew Neitsch

2016-04-26

Disclaimer

This is how people did things 2004–2007.

Disclaimer

This is how people did things 2004–2007.

It still works.

Disclaimer

This is how people did things 2004–2007.

It still works.

I think it's simpler and more efficient than the newer `pip`/`virtualenv` way.

Disclaimer

This is how people did things 2004–2007.

It still works.

I think it's simpler and more efficient than the newer `pip`/`virtualenv` way.

But since it's old and no longer popular, if you use it, some people will assume you don't what you're doing.

Package loading

```
import foo
```

Package loading

```
import foo
```

Check each entry in `sys.path` for a module

```
/usr/lib64/python2.7
~/.local/lib/python2.7/site-packages
/usr/lib64/python2.7/site-packages
/usr/lib/python2.7/site-packages
```

Package loading

```
import foo
```

Check each entry in `sys.path` for a module

- `foo/__init__.py`
- `foo.py`
- `foo.pyc`
- `foo.so`
- `...`

Package loading

```
import foo
```

To get specific library versions:

1. We can install a bunch of versions of a bunch of libraries in one place
   e.g., `site-packages/foo-1.7`, `site-packages/foo-1.8`

Package loading

```
import foo
```

To get specific library versions:

1. We can install a bunch of versions of a bunch of libraries in one place
2. Edit `sys.path` before importing anything we care about the version of e.g., `site-packages/foo-1.7`

Package loading

```
import foo
```

To get specific library versions:

1. We can install a bunch of versions of a bunch of libraries in one place
2. Edit `sys.path` before importing anything we care about the version of
3. Automate step 2

Package loading

```
import foo
```

To get specific library versions:

1. We can install a bunch of versions of a bunch of libraries in one place
2. Edit `sys.path` before importing anything we care about the version of
3. Automate step 2

That's `easy_install` and `pkg_resources`

Install the latest Django

```
$ easy_install --user Django
Searching for Django
Best match: Django 1.9.5
...
$ python -c 'import django; print django.VERSION'
(1, 9, 5, 'final', 0)
```

Install the latest Django

```
$ easy_install --user Django
Searching for Django
Best match: Django 1.9.5
...
$ python -c 'import django; print django.VERSION'
(1, 9, 5, 'final', 0)
```

Great! But what about that website still on Django 1.8.x?

Install the latest Django 1.8

```
$ easy_install --user 'Django >= 1.8, < 1.8.99'
Searching for Django>=1.8,<1.8.99
Best match: Django 1.8.12
...
$ python -c 'import django; print django.VERSION'
(1, 8, 12, 'final', 0)
```

Install the latest Django 1.8

```
$ easy_install --user 'Django >= 1.8, < 1.8.99'
Searching for Django>=1.8,<1.8.99
Best match: Django 1.8.12
...
$ python -c 'import django; print django.VERSION'
(1, 8, 12, 'final', 0)
```

The 1.8.99 is because 1.9rc1 $<$ 1.9

Install the latest Django 1.8

```
$ easy_install --user 'Django >= 1.8, < 1.8.99'
Searching for Django>=1.8,<1.8.99
Best match: Django 1.8.12
...
$ python -c 'import django; print django.VERSION'
(1, 8, 12, 'final', 0)
```

By default, you get the last version you installed.

But a project requires 1.9 ...

```
$ cat foo3.py

__requires__ = 'Django > 1.9, < 1.9.99'

import pkg_resources

import django
print django.VERSION
$ python foo3.py
(1, 9, 5, 'final', 0)
```

pkg_resources

We can dynamically define our version constraints in `__requires__`, and let Python adjust `sys.path` to point to installed package versions that satisfy those constraints.

Dynamic version selection

```
python_version = sys.argv[1]

__requires__ = 'Django >= {}, < {}.99'.format(
    python_version, python_version)

import pkg_resources

import django
print django.VERSION
print sys.modules['django']
```

```
python_version = sys.argv[1]

__requires__ = 'Django >= {}, < {}.99'.format(
    python_version, python_version)

def write_path(filename):
    with open(filename, 'w') as out:
        out.write("\n".join(sys.path + [""]))

write_path('orig_path')
import pkg_resources
write_path('new_path')

subprocess.call('diff -U99 orig_path new_path'.split())

import django
print django.VERSION
print sys.modules['django']
```

```
$ python foo.py 1.8
(1, 8, 12, 'final', 0)
<module 'django' from '~/.local/lib/python2.7/
   site-packages/Django-1.8.12-py2.7.egg/django/__init__.pyc'>
$ python foo.py 1.9
--- orig_path
+++ new_path
@@ -1,11 +1,12 @@
+~/.local/lib/python2.7/site-packages/Django-1.9.5-py2.7.egg
  ~
 ~/.local/lib/python2.7/site-packages/Django-1.8.12-py2.7.egg
  /usr/lib64/python27.zip
  /usr/lib64/python2.7
  /usr/lib64/python2.7/plat-linux2
  /usr/lib64/python2.7/lib-tk
  /usr/lib64/python2.7/lib-old
  /usr/lib64/python2.7/lib-dynload
  ~/.local/lib/python2.7/site-packages
  /usr/lib64/python2.7/site-packages
  /usr/lib/python2.7/site-packages
(1, 9, 5, 'final', 0)
<module 'django' from '/home/andrew/.local/lib/python2.7/
   site-packages/Django-1.9.5-py2.7.egg/django/__init__.pyc'>
```

Possible problem—

We have two Django versions in `sys.path`.
If we try to import a Django 1.8 module that was removed in 1.9, it might succeed.

Possible problem—

We have two Django versions in `sys.path`. If we try to import a Django 1.8 module that was removed in 1.9, it might succeed.

```
$ cat ~/.local/lib/python2.7/site-packages/easy-install.pth
import sys; sys.__plen = len(sys.path)
./Django-1.8.12-py2.7.egg
import sys; new=sys.path[sys.__plen:]; del sys.path[sys.__plen:];
    p=getattr(sys,'__egginsert',0); sys.path[p:p]=new;
    sys.__egginsert = p+len(new)
```

Possible problem—

We have two Django versions in `sys.path`. If we try to import a Django 1.8 module that was removed in 1.9, it might succeed.

```
$ cat ~/.local/lib/python2.7/site-packages/easy-install.pth
import sys; sys.__plen = len(sys.path)
./Django-1.8.12-py2.7.egg
import sys; new=sys.path[sys.__plen:]; del sys.path[sys.__plen:];
    p=getattr(sys,'__egginsert',0); sys.path[p:p]=new;
    sys.__egginsert = p+len(new)
```

You can just delete that.

Possible problem—

We have two Django versions in `sys.path`. If we try to import a Django 1.8 module that was removed in 1.9, it might succeed.

```
$ cat ~/.local/lib/python2.7/site-packages/easy-install.pth
import sys; sys.__plen = len(sys.path)
./Django-1.8.12-py2.7.egg
import sys; new=sys.path[sys.__plen:]; del sys.path[sys.__plen:];
    p=getattr(sys,'__egginsert',0); sys.path[p:p]=new;
    sys.__egginsert = p+len(new)
```

Or use the `easy_install -m` option.

Want version 1.8?

```
$ python foo.py 1.8
--- orig_path
+++ new_path
@@ -1,10 +1,11 @@
 ~
 /usr/lib64/python27.zip
 /usr/lib64/python2.7
 /usr/lib64/python2.7/plat-linux2
 /usr/lib64/python2.7/lib-tk
 /usr/lib64/python2.7/lib-old
 /usr/lib64/python2.7/lib-dynload
+~/.local/lib/python2.7/site-packages/Django-1.8.12-py2.7.egg
 ~/.local/lib/python2.7/site-packages
 /usr/lib64/python2.7/site-packages
 /usr/lib/python2.7/site-packages
(1, 8, 12, 'final', 0)
<module 'django' from '~/.local/lib/python2.7/site-packages/
    Django-1.8.12-py2.7.egg/django/__init__.pyc'>
```

Want version 1.9?

```
$ python foo.py 1.9
--- orig_path
+++ new_path
@@ -1,10 +1,11 @@
 ~
 /usr/lib64/python27.zip
 /usr/lib64/python2.7
 /usr/lib64/python2.7/plat-linux2
 /usr/lib64/python2.7/lib-tk
 /usr/lib64/python2.7/lib-old
 /usr/lib64/python2.7/lib-dynload
+~/.local/lib/python2.7/site-packages/Django-1.9.5-py2.7.egg
 ~/.local/lib/python2.7/site-packages
 /usr/lib64/python2.7/site-packages
 /usr/lib/python2.7/site-packages
(1, 9, 5, 'final', 0)
<module 'django' from '~/.local/lib/python2.7/site-packages/
    Django-1.9.5-py2.7.egg/django/__init__.pyc'>
```

Don't care?

```
$ python -c 'import django; print django.VERSION'
```

Don't care?

```
$ python -c 'import django; print django.VERSION'

Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named django
```

Please be more specific!

Conclusion

- `easy_install` installs to `site-packages/<module>-<vers>.egg` instead of `site-packages/<module>`, allowing multiple versions to be installed

Conclusion

- `easy_install` installs to `site-packages/<module>-<vers>.egg` instead of `site-packages/<module>`, allowing multiple versions to be installed
- `pkg_resources` dynamically adjusts `sys.path` based on `__requires__`

Conclusion

- `easy_install` installs to `site-packages/<module>-<vers>.egg` instead of `site-packages/<module>`, allowing multiple versions to be installed
- `pkg_resources` dynamically adjusts `sys.path` based on `__requires__`
- Edit or remove default library version in `easy-install.pth`

Conclusion

- `easy_install` installs to `site-packages/<module>-<vers>.egg` instead of `site-packages/<module>`, allowing multiple versions to be installed
- `pkg_resources` dynamically adjusts `sys.path` based on `__requires__`
- Edit or remove default library version in `easy-install.pth`
- Re-run `easy_install` to change default version for scripts

Comparison to pip

```
[andrew@localhost ~]$ pip install --user Django==1.6
Collecting Django==1.6
  Downloading Django-1.6-py2.py3-none-any.whl (6.7MB)
    100% |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX| 6.7MB 223kB/s
Installing collected packages: Django
Successfully installed Django-1.6

[andrew@localhost ~]$ pip install --user Django==1.7
Collecting Django==1.7
  Downloading Django-1.7-py2.py3-none-any.whl (7.4MB)
    100% |XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX| 7.4MB 112kB/s
Installing collected packages: Django
  Found existing installation: Django 1.6
    Uninstalling Django-1.6:
      Successfully uninstalled Django-1.6
```

Other features:

- Metadata in `EGG-INFO` directory
- Recursive dependency resolution
- API for loading data files from modules, even if module is zipped

Other features:

- Metadata in `EGG-INFO` directory
- Recursive dependency resolution
- API for loading data files from modules, even if module is zipped

Docs:

- peak.telecommunity.com/DevCenter/EasyInstall
- peak.telecommunity.com/DevCenter/PkgResources