Disclaimer about this talk

- pip only has one "API": its Command Line Interface
Disclaimer about this talk

- pip only has one "API": its Command Line Interface
- so internals are subjects to changes
Disclaimer about this talk

- pip only has one "API": its Command Line Interface
- so internals are subjects to changes
- corresponds to pip 7.1.0 code
Outline for section 1

1. How does pip resolve dependencies?
2. How does pip perform an installation?
3. How does pip select the packages?
1 - How does pip resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my_package, https://foo.com/bar-1.0.0.tar.gz, ./my_package)
1 - How does pip resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my_package, https://foo.com/bar-1.0.0.tar.gz, ./my_package)
- -e/--editable editable_package
1 - How does *pip* resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my-package, https://foo.com/bar-1.0.0.tar.gz, ./my-package)
- `-e/--editable` editable package
- `-r/--requirement` requirements.txt
1 - How does pip resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my_package, https://foo.com/bar-1.0.0.tar.gz, ./my_package)
- `-e`/`--editable` editable_package
- `-r`/`--requirement` requirements.txt
- `-c`/`--constraint` constraints.txt (since 7.1)
1 - How does pip resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my_package, https://foo.com/bar-1.0.0.tar.gz, ./my_package)
- -e/--editable editable_package
- -r/--requirement requirements.txt
- -c/--constraint constraints.txt (since 7.1)
1 - How does pip resolve dependencies?

**pip.req_set.RequirementSet**

- direct args (my_package, https://foo.com/bar-1.0.0.tar.gz, ./my_package)
- -e/--editable editable_package
- -r/--requirement requirements.txt
- -c/--constraint constraints.txt (since 7.1)

**Or a combination**

```
pip install -e ./pkg1 django -e ./pkg2 -r reqs.txt -c constraint.txt
```
How does pip resolve dependencies?

```
RequirementCommand.populate_requirement_set

- from command options
```
1 - How does pip resolve dependencies?

RequirementCommand.populate_requirement_set
- from command options
- calls RequirementSet.add_requirement

Xavier Fernandez (Polyconseil)
RequirementCommand.populate_requirement_set

- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files
1 - How does pip resolve dependencies?

RequirementCommand.populate_requirement_set

- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files
1 - How does pip resolve dependencies?

**RequirementCommand.populate_requirement_set**
- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files

**RequirementSet.add_requirement**
- checks environment markers (e.g. python_version==‘2.6’, etc)
1 - How does pip resolve dependencies?

Environment markers

- defined in PEP345 and extended in PEP426 (draft)
1 - How does pip resolve dependencies?

Environment markers

- defined in PEP345 and extended in PEP426 (draft)
- allows to specify required dependencies on specific environment
Environment markers

- defined in PEP345 and extended in PEP426 (draft)
- allows to specify required dependencies on specific environment
1 - How does pip resolve dependencies?

Environment markers
- defined in PEP345 and extended in PEP426 (draft)
- allows to specify required dependencies on specific environment

Examples
- `extras_require={':python_version=="3.3"': ['asyncio']}`
Environment markers

- defined in PEP345 and extended in PEP426 (draft)
- allows to specify required dependencies on specific environment

Examples

- `extras_require={':python_version=='3.3'': ['asyncio']}
- `extras_require={ 'signatures': ['keyring'], 'signatures:sys_platform!='win32'': ['pyxdg'] }`
1 - How does pip resolve dependencies?

RequirementCommand.populate_requirement_set

- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files

RequirementSet.add_requirement

- checks environment markers (e.g. python_version==’2.6’, etc)
1 - How does pip resolve dependencies?

**RequirementCommand.populate_requirement_set**
- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files

**RequirementSet.add_requirement**
- checks environment markers (e.g. python_version==’2.6’, etc)
- if already there and user supplied, reject any double requirement
1 - How does pip resolve dependencies?

**RequirementCommand.populate_requirement_set**
- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files

**RequirementSet.add_requirement**
- checks environment markers (e.g. `python_version==’2.6’, etc`)
- if already there and user supplied, reject any double requirement
- **except** if previous one is a constraint, then use constraint version
1 - How does pip resolve dependencies?

```
Constraint
$ echo "django==1.8.3" > constraint.txt
$ pip install django==1.4.3 -c constraint.txt
Successfully installed django-1.8.3
```
How does pip resolve dependencies?

1. **RequirementCommand.populate_requirement_set**
   - from command options
   - calls `RequirementSet.add_requirement`
   - constraints, args, editables, requirement files

2. **RequirementSet.add_requirement**
   - checks environment markers (e.g. `python_version==\'2.6\',` etc)
   - if already there and user supplied, reject any double requirement
   - **except** if previous one is a constraint, then use constraint version
1 - How does pip resolve dependencies?

**RequirementCommand.populate_requirement_set**

- from command options
- calls RequirementSet.add_requirement
- constraints, args, editables, requirement files

**RequirementSet.add_requirement**

- checks environment markers (e.g. `python_version==’2.6’, etc)
- if already there and user supplied, reject any double requirement
- except if previous one is a constraint, then use constraint version
- if already there and comes from dependency, ignore
1 - How does pip resolve dependencies?

Conflict between user and dependency

- package simple with install_requires=['pep8==1.2']
1 - How does pip resolve dependencies?

Conflict between user and dependency

- package simple with install_requires= ['pep8==1.2']
- pip install simple pep8==1.3
1 - How does pip resolve dependencies?

Conflict between user and dependency

- package simple with install_requires=[‘pep8==1.2’]
- pip install simple pep8==1.3
- add_requirement(‘simple’) and add_requirement(pep8==1.3)
1 - How does pip resolve dependencies?

Conflicts between user and dependency

- package `simple` with install_requires=[‘pep8==1.2’]
- `pip install simple pep8==1.3`
- `add_requirement(‘simple’) and add_requirement(pep8==1.3)`
- `add_requirement(pep8==1.2)` ignored
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**

- calls RequirementSet._prepare_file
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**

- calls RequirementSet._prepare_file
- on unnamed requirements
1 - How does pip resolve dependencies?

```
RequirementSet.prepare_files

- calls RequirementSet._prepare_file
- on unnamed requirements
- named requirements
```
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**

- calls `RequirementSet._prepare_file`
- on unnamed requirements
- named requirements
- newly found dependencies
### 1 - How does pip resolve dependencies?

#### RequirementSet.prepare_files

- Calls `RequirementSet._prepare_file`
- On unnamed requirements
- Named requirements
- Newly found dependencies

- Check if already installed and if new version needed
- For non-link reqs, use finder.find requirement to populate req.link
- Calls req.link.setter and search for a cached wheel for this link
- Download/update/unpack source files
- Compute metadata: setup.py, egg info, or directly from wheel
- Compute dependencies (with extras)
1 - How does pip resolve dependencies?

RequirementSet.prepare_files
- calls RequirementSet._prepare_file
- on unnamed requirements
- named requirements
- newly found dependencies

RequirementSet._prepare_file
- check if already installed and if new version needed
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**
- calls `RequirementSet._prepare_file`
- on unnamed requirements
- named requirements
- newly found dependencies

**RequirementSet._prepare_file**
- check if already installed and if new version needed
- for non-link reqs, use `finder.find_requirement` to populate `req.link`
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**
- calls `RequirementSet._prepare_file`
- on unnamed requirements
- named requirements
- newly found dependencies

**RequirementSet._prepare_file**
- check if already installed and if new version needed
- for non-link reqs, use `finder.find_requirement` to populate `req.link`
- calls `req.link.setter` and search for a cached wheel for this link
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**
- calls `RequirementSet._prepare_file`
- on unnamed requirements
- named requirements
- newly found dependencies

**RequirementSet._prepare_file**
- check if already installed and if new version needed
- for non-link reqs, use `finder.find_requirement` to populate `req.link`
- calls `req.link.setter` and search for a cached wheel for this link
- download/update/unpack source files
### RequirementSet.prepare_files
- calls RequirementSet._prepare_file
- on unnamed requirements
- named requirements
- newly found dependencies

### RequirementSet._prepare_file
- check if already installed and if new version needed
- for non-link reqs, use finder.find_requirement to populate req.link
- calls req.link.setter and search for a cached wheel for this link
- download/update/unpack source files
- compute metadata: setup.py egg_info or directly from wheel
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**
- calls RequirementSet._prepare_file
- on unnamed requirements
- named requirements
- newly found dependencies

**RequirementSet._prepare_file**
- check if already installed and if new version needed
- for non-link reqs, use finder.find_requirement to populate req.link
- calls req.link.setter and search for a cached wheel for this link
- download/update/unpack source files
- compute metadata: setup.py egg_info or directly from wheel
- compute dependencies (with extras)
1 - How does pip resolve dependencies?

```
RequirementSet.prepare_files

- pip install flake8
```
1 - How does pip resolve dependencies?

```python
RequirementSet.prepare_files

- pip install flake8
- adds pep8, pyflakes, mccabe
```
How does pip resolve dependencies?

RequirementSet.prepare_files

- pip install flake8
- adds pep8, pyflakes, mccabe
- populate links for all
1 - How does pip resolve dependencies?

**RequirementSet.prepare_files**

- `pip install flake8`
- adds pep8, pyflakes, mccabe
- populate links for all
- at the end, in a temporary directory (build_location)
1 - How does pip resolve dependencies?

RequirementSet.prepare_files

- pip install flake8
- adds pep8, pyflakes, mccabe
- populate links for all
- at the end, in a temporary directory (build_location)
- 4 directories, one for each dependency
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)
- if `wheel` package available
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir
Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options `--download` or `--no-cache-dir`

Exceptions

- already wheels
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir

Exceptions

- already wheels
- editable requirements / requirements pointing to a VCS
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir

Exceptions

- already wheels
- editable requirements / requirements pointing to a VCS
- --no-binary was used with :all:/the specific package
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir

Exceptions

- already wheels
- editable requirements / requirements pointing to a VCS
- --no-binary was used with :all:/the specific package
- other special cases
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)
- if wheel package available
- if not using options --download or --no-cache-dir

Exceptions
- already wheels
- editable requirements / requirements pointing to a VCS
- --no-binary was used with :all:/the specific package
- other special cases
1 - How does pip resolve dependencies?

**Optional Wheel caching (pip 7+)**
- if wheel package available
- if not using options --download or --no-cache-dir

**Exceptions**
- already wheels
- editable requirements / requirements pointing to a VCS
- --no-binary was used with :all:/the specific package
- other special cases

**Wheel caching**
- on success, update link to point on new wheel file and unpack the wheel in the build location
1 - How does pip resolve dependencies?

Optional Wheel caching (pip 7+)

- if wheel package available
- if not using options --download or --no-cache-dir

Exceptions

- already wheels
- editable requirements / requirements pointing to a VCS
- --no-binary was used with :all:/the specific package
- other special cases

Wheel caching

- on success, update link to point on new wheel file and unpack the wheel in the build_location
- else, no issue
Outline for section 2

1. How does pip resolve dependencies?
2. How does pip perform an installation?
3. How does pip select the packages?
2 - How does pip perform an installation?

Wheels

- already unpacked in prepare_files
2 - How does pip perform an installation?

Wheels

- already unpacked in prepare_files
- move directory to site-packages
2 - How does pip perform an installation?

Wheels

- already unpacked in prepare_files
- move directory to site-packages
- some complexities for scripts/data
2 - How does pip perform an installation?

<table>
<thead>
<tr>
<th>Wheels</th>
</tr>
</thead>
<tbody>
<tr>
<td>• already unpacked in prepare_files</td>
</tr>
<tr>
<td>• move directory to site-packages</td>
</tr>
<tr>
<td>• some complexities for scripts/data</td>
</tr>
<tr>
<td>• RECORD file</td>
</tr>
</tbody>
</table>
2 - How does pip perform an installation?

Wheels
- already unpacked in prepare_files
- move directory to site-packages
- some complexities for scripts/data
- RECORD file
2 - How does pip perform an installation?

Wheels
- already unpacked in prepare_files
- move directory to site-packages
- some complexities for scripts/data
- RECORD file

Note
No setuptools needed
setup.py install

- req.install.InstallRequirement.install
2 - How does pip perform an installation?

setup.py install

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`
setup.py install

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`
- `python setup.py install` with some options, mainly:
2 - How does pip perform an installation?

setup.py install

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`
- `python setup.py install with some options, mainly:
  - `--record tmp_path/installed-files.txt`
2 - How does pip perform an installation?

**setup.py install**

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`  
- `python setup.py install` with some options, mainly:
  - `--record tmp/path/installed-files.txt`
  - `--single-version-externally-managed`
2 - How does pip perform an installation?

setup.py install

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`
- `python setup.py install with some options, mainly:
  - --record tmp_path/installed-files.txt`
  - `--single-version-externally-managed`
2 - How does pip perform an installation?

**setup.py install**

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))`  
  
- python setup.py install *with some options*, mainly:
  - `--record tmp_path/installed-files.txt`
  - `--single-version-externally-managed`

**Avoid python setup.py install**

- *with setuptools: easy_install invocation*
2 - How does pip perform an installation?

**setup.py install**

- `req_install.InstallRequirement.install`
- `import setuptools; exec(compile(open(’setup.py’))`
- `python setup.py install with some options, mainly:`
- `--record tmp_path/installed-files.txt`
- `--single-version-externally-managed`

**Avoid python setup.py install**

- with setuptools: easy_install invocation
- with distutils: no installed-files.txt
2 - How does pip perform an installation?

**setup.py install**

- `reqinstall.InstallRequirement.install`
- `import setuptools; exec(compile(open('setup.py'))
- `python setup.py install` with some options, mainly:
  - `--record tmp_path/installed-files.txt`
  - `--single-version-externally-managed`

**Avoid python setup.py install**

- with setuptools: easy_install invocation
- with distutils: no installed-files.txt
- prefer pip install.
How does pip perform an installation?

req_install.InstallRequirement.install_editable
2 - How does pip perform an installation?

- `req_install.InstallRequirement.install_editable`
- `import setuptools; exec(compile(open('setup.py'))`
How does pip perform an installation?

- `req_install.InstallRequirement.install_editable`
- `import setuptools; exec(compile(open('setup.py')))
- `python setup.py develop --no-deps`
2 - How does pip perform an installation?

Rollbacks

- uninstalls old version first
2 - How does pip perform an installation?

Rollbacks

- uninstalls old version first
- tries to install
Rollbacks

- uninstalls old version first
- tries to install
- on success, all good, commits uninstall
2 - How does pip perform an installation?

Rollbacks
- uninstalls old version first
- tries to install
- on success, all good, commits uninstall
- else, rollbacks uninstall
2 - How does pip perform an installation?

Rollbacks
- uninstalls old version first
- tries to install
- on success, all good, commits uninstall
- else, rollbacks uninstall
2 - How does pip perform an installation?

Rollbacks
- Uninstalls old version first
- Tries to install
- On success, all good, commits uninstall
- Else, rollbacks uninstall

Only one rollback
- Rollbacks concerns the last uninstall
2 - How does pip perform an installation?

**Rollbacks**

- Uninstalls old version first
- Tries to install
- On success, all good, commits uninstall
- Else, rollbacks uninstall

**Only one rollback**

- Rollbacks concerns the last uninstall
- Not the whole install call
2 - How does pip perform an installation?

Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
2 - How does pip perform an installation?

Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
- A1 and B1 in your venv
2 - How does pip perform an installation?

Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
- A1 and B1 in your venv
- pip install A==2.0
2 - How does pip perform an installation?

Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
- A1 and B1 in your venv
- `pip install A==2.0`
- it might: uninstall B1 and install B2 successfully

Xavier Fernandez (Polyconseil)
Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
- A1 and B1 in your venv
- pip install A==2.0
- it might: uninstall B1 and install B2 successfully
- but then might uninstall A1 and crash on A2 install
2 - How does pip perform an installation?

Rollbacks

- `A==1.0` requires `B==1.0` and `A==2.0` requires `B==2.0`
- `A1` and `B1` in your `venv`
- `pip install A==2.0`
- It might: uninstall `B1` and install `B2` successfully
- But then might uninstall `A1` and crash on `A2` install
- `A1` will be restored
Rollbacks

- A==1.0 requires B==1.0 and A==2.0 requires B==2.0
- A1 and B1 in your venv
- pip install A==2.0
- it might: uninstall B1 and install B2 successfully
- but then might uninstall A1 and crash on A2 install
- A1 will be restored
- but you end up with A1 and B2
Installation order

- tries to respect the provided (from args) order
Installation order

- tries to respect the provided (from args) order
- but installs dependencies first
2 - How does pip perform an installation?

**Installation order**

- tries to respect the provided (from args) order
- but installs dependencies first
## 2 - How does pip perform an installation?

### Installation order
- tries to respect the provided (from args) order
- but installs dependencies first

### Implementation detail
- No order guarantee
2 - How does pip perform an installation?

Installation order
- tries to respect the provided (from args) order
- but installs dependencies first

Implementation detail
- No order guarantee
- if you need this order, use `setup_requires`
### Installation order
- tries to respect the provided (from args) order
- but installs dependencies first

### Implementation detail
- No order guarantee
- if you need this order, use `setup_requires`
Installation order

- tries to respect the provided (from args) order
- but installs dependencies first

Implementation detail

- No order guarantee
- if you need this order, use `setup_requires`

setup_requires

- currently "owned" by setupools
2 - How does pip perform an installation?

Installation order
- tries to respect the provided (from args) order
- but installs dependencies first

Implementation detail
- No order guarantee
- if you need this order, use setup_requires

setup_requires
- currently "owned" by setup tools
- might call easy_install
2 - How does pip perform an installation?

**Installation order**
- tries to respect the provided (from args) order
- but installs dependencies first

**Implementation detail**
- No order guarantee
- if you need this order, use `setup_requires`

**setup_requires**
- currently "owned" by setuptools
- might call `easy_install`
- hopefully, pip might take control of this feature
Outline for section 3

1. How does pip resolve dependencies?
2. How does pip perform an installation?
3. How does pip select the packages?
3 - How does pip select the packages?

<table>
<thead>
<tr>
<th>Context</th>
<th>pip install foo_bar</th>
</tr>
</thead>
</table>

- different versions
- different formats
- different locations

Xavier Fernandez (Polyconseil)  pip internals  July 20, 2015  22 / 38
3 - How does pip select the packages?

Context

```bash
pip install foo_bar
```

- https://some_pypi.com/simple/foo_bar/foo_bar-1.2.tar.gz
- https://some_pypi.com/simple/foo_bar/foo_bar-1.0-py3-none-any.whl
- /home/user/wheelhouse/foo_bar-3.0-py3-none-any.whl
- dozens of others...
3 - How does pip select the packages?

Context

```
pip install foo_bar
```

- https://some_pypi.com/simple/foo_bar/foo_bar-1.2.tar.gz
- https://some_pypi.com/simple/foo_bar/foo_bar-1.0-py3-none-any.whl
- /home/user/wheelhouse/foo_bar-3.0-py3-none-any.whl
- dozens of others...

What file should pip use?

- different versions
3 - How does pip select the packages?

Context

```
pip install foo_bar
```

- `https://some_pypi.com/simple/foo_bar/foo_bar-1.2.tar.gz`
- `https://some_pypi.com/simple/foo_bar/foo_bar-1.0-py3-none-any.whl`
- `/home/user/wheelhouse/foo_bar-3.0-py3-none-any.whl`
- dozens of others...

What file should pip use?

- different versions
- different formats
3 - How does pip select the packages?

**Context**

```
pip install foo_bar
```

- https://some_pypi.com/simple/foo_bar/foo_bar-1.2.tar.gz
- https://some_pypi.com/simple/foo_bar/foo_bar-1.0-py3-none-any.whl
- /home/user/wheelhouse/foo_bar-3.0-py3-none-any.whl
- dozens of others...

**What file should pip use?**

- different versions
- different formats
- different locations
Where the magic happens: pip/index.py

- `PackageFinder.find_requirement(req, upgrade)`
3 - How does pip select the packages?

Where the magic happens: pip/index.py

- `PackageFinder.find_requirement(req, upgrade)`
- `req` is an `InstallRequirement`: `django`, `django==1.8.3`, `Django<1.8`
3 - How does pip select the packages?

Where the magic happens: pip/index.py

- `PackageFinder.find_requirement(req, upgrade)`
- `req` is an `InstallRequirement`: `django`, `django==1.8.3`, `Django<1.8`
- `upgrade` is a boolean (``-upgrade option``)
3 - How does pip select the packages?

Where the magic happens: pip/index.py

- `PackageFinder.find_requirement(req, upgrade)`
- `req` is an `InstallRequirement`: `django`, `django==1.8.3`, `Django<1.8`
- `upgrade` is a boolean (`--upgrade` option)
- Returns an `InstallationCandidate` (a file/location)
3 - How does pip select the packages?

```python
find_requirement(req, upgrade)

- calls `self._find_all_versions(req.name)`
```
3 - How does pip select the packages?

```python
find_requirement(req, upgrade)
```

- calls `self._find_all_versions(req.name)`
- for django on PyPI, 112 InstallationCandidate
3 - How does pip select the packages?

find_requirement(req, upgrade)

- calls `self._find_all_versions(req.name)`
- for `django` on PyPI, 112 `InstallationCandidate`
- on my setup (with private mirror and wheelhouse), 249
3 - How does pip select the packages?

```python
find_requirement(req, upgrade)
```
- calls `self._find_all_versions(req.name)`
- for `django` on PyPI, 112 `InstallationCandidate`
- on my setup (with private mirror and wheelhouse), 249
- filter on version requirements (==1.8.3) and `--pre` option
3 - How does pip select the packages?

```python
find_requirement(req, upgrade)
```

- calls `self.find_all_versions(req.name)`
- for `django` on PyPI, 112 `InstallationCandidate`
- on my setup (with private mirror and wheelhouse), 249
- filter on version requirements (`==1.8.3`) and `--pre` option
- (stable) sort by version
3 - How does pip select the packages?

```python
find_requirement(req, upgrade)
```

- calls `self._find_all_versions(req.name)`
- for `django` on PyPI, 112 `InstallationCandidate`
- on my setup (with private mirror and wheelhouse), 249
- filter on version requirements (`==1.8.3`) and `--pre` option
- (stable) sort by version
- returns most recent/currently installed version (`--upgrade` option)
3 - How does pip select the packages?

_find_all_versions: Three main sources

- indexes: --index-url, --extra-index-url or --no-index options
3 - How does pip select the packages?

_find_all_versions: Three main sources

- indexes: --index-url, --extra-index-url or --no-index options
- find_links: --find-links options
3 - How does pip select the packages?

_find_all_versions: Three main sources

- indexes: `--index-url`, `--extra-index-url` or `--no-index` options
- find_links: `--find-links` options
- dependency_links: provided by some setup.py if `--process-dependency-links`
3 - How does pip select the packages?

Indexes

- expects something like scheme://some_index/
3 - How does pip select the packages?

Indexes

- expects something like scheme://some_index/
- goes to scheme://some_index/package_name to list available files
3 - How does pip select the packages?

Indexes

- expects something like scheme://some_index/
- goes to scheme://some_index/package_name to list available files
## 3 - How does pip select the packages?

### Indexes
- expects something like scheme:////some_index/
- goes to scheme:////some_index/package_name to list available files

### Normalized name
- python-dateutil
3 - How does pip select the packages?

Indexes
- expects something like `scheme://some_index/`
- goes to `scheme://some_index/package_name` to list available files

Normalized name
- python-dateutil
- Python-Dateutil
3 - How does pip select the packages?

Indexes
- expects something like scheme://some_index/
- goes to scheme://some_index/package_name to list available files

Normalized name
- python-dateutil
- Python-Dateutil
- python_dateutil
3 - How does pip select the packages?

Indexes
- expects something like scheme://some_index/
- goes to scheme://some_index/package_name to list available files

Normalized name
- python-dateutil
- Python-Dateutil
- python_dateutil
- pYthOn_datEutiL
3 - How does pip select the packages?

With modern indexes - PyPI

- GET on https://pypi/simple/Python_Dateutil
3 - How does pip select the packages?

With modern indexes - PyPI

- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
3 - How does pip select the packages?

With modern indexes - PyPI

- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/
3 - How does pip select the packages?

With modern indexes - PyPI

- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/
3 - How does pip select the packages?

**With modern indexes - PyPI**
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/

**Older/simpler indexes**
- GET on https://pypi/simple/Python_Dateutil returns a 404
3 - How does pip select the packages?

**With modern indexes - PyPI**
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/

**Older/simpler indexes**
- GET on https://pypi/simple/Python_Dateutil returns a 404
- try GET on https://pypi/simple/ - (2,7M for PyPI)
3 - How does pip select the packages?

<table>
<thead>
<tr>
<th>With modern indexes - PyPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>• GET on <a href="https://pypi/simple/Python_Dateutil">https://pypi/simple/Python_Dateutil</a></td>
</tr>
<tr>
<td>• redirects to <a href="https://pypi/simple/python-dateutil">https://pypi/simple/python-dateutil</a></td>
</tr>
<tr>
<td>• re-directs to <a href="https://pypi/simple/python-dateutil/">https://pypi/simple/python-dateutil/</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Older/simpler indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• GET on <a href="https://pypi/simple/Python_Dateutil">https://pypi/simple/Python_Dateutil</a> returns a 404</td>
</tr>
<tr>
<td>• try GET on <a href="https://pypi/simple/">https://pypi/simple/</a> - (2.7M for PyPI)</td>
</tr>
<tr>
<td>• match all the links of the index on the normalized form</td>
</tr>
</tbody>
</table>
3 - How does pip select the packages?

With modern indexes - PyPI
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-directs to https://pypi/simple/python-dateutil/

Older/simpler indexes
- GET on https://pypi/simple/Python_Dateutil returns a 404
- try GET on https://pypi/simple/ - (2,7M for PyPI)
- match all the links of the index on the normalized form
- Note: this is only performed on the first index but used on all
3 - How does pip select the packages?

### With modern indexes - PyPI
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/

### Older/simpler indexes
- GET on https://pypi/simple/Python_Dateutil returns a 404
- try GET on https://pypi/simple/ - (2.7M for PyPI)
- match all the links of the index on the normalized form
- Note: this is only performed on the first index but used on all
3 - How does pip select the packages?

**With modern indexes - PyPI**
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-redirects to https://pypi/simple/python-dateutil/

**Older/simpler indexes**
- GET on https://pypi/simple/Python_Dateutil returns a 404
- try GET on https://pypi/simple/ - (2,7M for PyPI)
- match all the links of the index on the normalized form
- Note: this is only performed on the first index but used on all

**New locations**
- https://pypi/simple/
3 - How does pip select the packages?

**With modern indexes - PyPI**
- GET on https://pypi/simple/Python_Dateutil
- redirects to https://pypi/simple/python-dateutil
- re-directs to https://pypi/simple/python-dateutil/

**Older/simpler indexes**
- GET on https://pypi/simple/Python_Dateutil returns a 404
- try GET on https://pypi/simple/ - (2.7M for PyPI)
- match all the links of the index on the normalized form
- Note: this is only performed on the first index but used on all

**New locations**
- https://pypi/simple/
- https://pypi/simple/python-dateutil
3 - How does pip select the packages?

indexes

\_find\_url\_name

index\_links find\_links dependency\_links
3 - How does pip select the packages?

![Diagram showing the process of pip selecting packages]

- Indexes
- `_find_url_name`
- `index_links`
- `find_links`
- `dependency_links`
- `_sort_locations`
- `file_locations`
- `url_locations`
3 - How does pip select the packages?

Diagram:
- indexes
  - _find_url_name
    - index_links
    - find_links
      - dependency_links
    - _sort_locations
      - file_locations
      - url_locations
3 - How does pip select the packages?

- indexes
- index_links
- find_url_name
- find_links
- dependency_links
- sort_locations
- file_locations
- url_locations
3 - How does pip select the packages?

```python
_sort_locations

if the scheme == file or if the path exists:
    if is a file:
        if html: -> urls
        else: -> files
    elif is a directory:
        if dealing with find_links:
            for all files:
                if html: -> urls
                else: -> files
        else: -> urls
    else: -> urls
else: -> urls
```
3 - How does pip select the packages?
3 - How does pip select the packages?

- indexes
  - _find_url_name
    - index_links
    - find_links
    - dependency_links

- _sort_locations
  - file_locations
  - url_locations
  - find_links_locations
3 - How does pip select the packages?

```
indexes
   ↓ _find_url_name
index_links  find_links  dependency_links
   ↓ _sort_locations
file_locations  url_locations  find_links_locations  dep_links_locations
```
3 - How does pip select the packages?

indexes
  ↓ _find_url_name
index_links | find_links | dependency_links
  ↓ _sort_locations
file_locations | url_locations | find_links_locations | dep_links_locations
  ↓ _validate_secure_origin
3 - How does pip select the packages?

_validate_secure_origin for url_locations

- protocol, hostname, port
- ("https", "*", "*")
- ("*", "localhost", "*")
- ("*", "127.0.0.0/8", "*")
- ("*", "::1/128", "*")
- ("file", "*", None)
3 - How does pip select the packages?

_validate_secure_origin for url_locations

- protocol, hostname, port
- ("https", "*", "*"),
- ("*", "localhost", "*"),
- ("*", "127.0.0.0/8", "*")
- ("*", "::1/128", "*")
- ("file", "*", None),
- ("*", host, "*") for all hosts from --trusted-host option
3 - How does pip select the packages?

- **indexes**
  - _find_url_name
    - index_links
    - find_links
    - dependency_links

- _sort_locations
  - file_locations
  - url_locations
  - find_links_locations
  - dep_links_locations

- _validate_secure_origin
3 - How does pip select the packages?

 indexes
  ↓ _find_url_name
index_links  ↓ find_links  ↓ dependency_links
  ↓ _sort_locations
file_locations  ↓ url_locations  ↓ find_links_locations  ↓ dep_links_locations
  ↓ _validate_secure_origin
  ↓ HTMLPage
3 - How does pip select the packages?

HTMLPage

- for urls looking like archives, check content type first
  if "text/html", continue
- get content and parse
- check all links
- if rel contains 'homepage' or 'download'
  add it (untrusted to url_locations)
### 3 - How does pip select the packages?

**HTMLPage**
- for urls looking like archives, check content type first if "text/html", continue
- get content and parse
- check all links
- if rel contains 'homepage' or 'download'
  add it (untrusted to url_locations)

**Note**
- with api_version 2 i.e. PyPI, `<a>` can be marked with rel="internal"
- only trust internals
- `<meta name="api-version" value="2" />` in the page head
- `--allow-external/--allow-all-external`
3 - How does pip select the packages?

```
indexes
  └── find_url_name
    ├── index_links
    │   └── _sort_locations
    │       ├── file_locations
    │       └── url_locations
    │           └── find_links_locations
    │               └── dep_links_locations
    │                   └── _validate_secure_origin
    │                           └── HTMLPage
    │                               └── HTMLPage
```
3 - How does pip select the packages?

Indexes

 indexes

_find_url_name

 index_links

 find_links

 dependency_links

_sort_locations

 file_locations

 url_locations

 find_links_locations

 dep_links_locations

_validate_secure_origin

 HTMLPage

 links

_package_versions
3 - How does pip select the packages?

- package versions filter
  - push egg links (containing #egg=) to the end
3 - How does pip select the packages?

**package versions filter**

- push egg links (containing #egg=) to the end
- select/reject on lots of criteria
3 - How does pip select the packages?

- package versions filter

- push egg links (containing #egg=) to the end
- select/reject on lots of criteria

Criteria:
- unsupported format (like bz2 on some install)
- macos10x files ending in .zip
- --only-binary/--no-binary options
- for wheel files: check compatibility and for non-windows/macos installs, refuse platform specific wheels from PyPI
- tries to extract version from filename and reject file if fails
- if version contains -py2/-py3/-py27/etc, check python version
- check internal/external hosted files and verifiable/unverifiable files
- --allow-external/--allow-all-external/--allow-unverified
3 - How does pip select the packages?

**package_versions filter**
- push egg links (containing `#egg=`) to the end
- select/reject on lots of criteria

**Criteria**
- unsupported format (like bz2 on some install)
3 - How does pip select the packages?

_package_versions filter

- push egg links (containing #egg=) to the end
- select/reject on lots of criteria

Criteria

- unsupported format (like bz2 on some install)
- macOS10x files ending in .zip
3 - How does pip select the packages?

**package_versions filter**
- push egg links (containing `#egg=`) to the end
- select/reject on lots of criteria

**Criteria**
- unsupported format (like bz2 on some install)
- macos10x files ending in .zip
- `--only-binary/--no-binary` options
3 - How does pip select the packages?

**package versions filter**
- push egg links (containing #egg=) to the end
- select/reject on lots of criteria

**Criteria**
- unsupported format (like bz2 on some install)
- macos10x files ending in .zip
- --only-binary/--no-binary options
- for wheel files: check compatibility and for non-windows/macOS installs, refuse platform specific wheels from PyPI
### 3 - How does pip select the packages?

<table>
<thead>
<tr>
<th>Package Versions Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>push egg links (containing #egg=) to the end</td>
</tr>
<tr>
<td>select/reject on lots of criteria</td>
</tr>
</tbody>
</table>

### Criteria

- unsupported format (like bz2 on some install)
- macos10x files ending in .zip
- `--only-binary/--no-binary` options
- for wheel files: check compatibility and for non-windows/macOS installs, refuse platform specific wheels from PyPI
- tries to extract version from filename and reject file if fails
3 - How does pip select the packages?

package versions filter

- push egg links (containing #egg=) to the end
- select/reject on lots of criteria

Criteria

- unsupported format (like bz2 on some install)
- macos10x files ending in .zip
- `--only-binary/--no-binary` options
- for wheel files: check compatibility and for non-windows/macos installs, refuse platform specific wheels from PyPI
- tries to extract version from filename and reject file if fails
- if version contains -py2/-py3/-py27/etc, check python version
3 - How does pip select the packages?

**package versions filter**
- push egg links (containing `#egg=`) to the end
- select/reject on lots of criteria

**Criteria**
- unsupported format (like bz2 on some install)
- macos10x files ending in `.zip`
- `--only-binary/--no-binary` options
- for wheel files: check compatibility and for non-windows/macos installs, refuse platform specific wheels from PyPI
- tries to extract version from filename and reject file if fails
- if version contains `-py2/-py3/-py27/etc`, check python version
- check internal/external hosted files and verifiable/unverifiable files
  `--allow-external/--allow-all-external/--allow-unverified`
3 - How does pip select the packages?

Preference Order

- file sources are preferred

Note: Wheel cache (pip 7 feature) has no impact on find all versions.
3 - How does pip select the packages?

Preference Order

- file sources are preferred
- then direct find_links files

Note: Wheel cache (pip 7 feature) has no impact on find all versions.
3 - How does pip select the packages?

Preference Order

- file sources are preferred
- then direct find_links files
- then urls links
3 - How does pip select the packages?

Preference Order

- file sources are preferred
- then direct find_links files
- then urls links
- direct dependency_links files
3 - How does pip select the packages?

Preference Order
- file sources are preferred
- then direct find_links files
- then urls links
- direct dependency_links files
- this is what _find_all_versions returns

Note: Wheel cache (pip 7 feature) has no impact on find all versions.
3 - How does pip select the packages?

Preference Order

- file sources are preferred
- then direct find_links files
- then urls links
- direct dependency_links files
- this is what _find_all_versions returns

Note: Wheel cache (pip 7 feature) has no impact on find all versions.
3 - How does pip select the packages?

Preference Order

- file sources are preferred
- then direct find_links files
- then urls links
- direct dependency_links files
- this is what _find_all_versions returns

Note

Wheel cache (pip 7 feature) has no impact on _find_all_versions
3 - How does pip select the packages?

```python
-find_all_versions('pep8')

[InstallationCandidate('pep8', Version('1.5.7'),
   Link file:///home/user/wheelhouse/pep8-1.5.7-py2.py3-none-any.whl),
InstallationCandidate('pep8', Version('1.5.0'),
   Link file:///home/user/wheelhouse/pep8-1.5.0-py27-none-any.whl),
...,
InstallationCandidate('pep8', Version('0.3.1'),
   Link https://private_pypi/simple/pep8/pep8-0.3.1.tar.gz),
InstallationCandidate('pep8', Version('0.4.1'),
   Link https://private_pypi/simple/pep8/pep8-0.4.1.tar.gz),
...,
InstallationCandidate('pep8', Version('1.5.2'),
   Link https://pypi.python.org/packages/3.3/p/pep8/
   pep8-1.5.2-py2.py3-none-any.whl#md5=a0caba277a2f491b1634246a338a1235),
InstallationCandidate('pep8', Version('1.5.3'),
   Link https://pypi.python.org/packages/3.3/p/pep8/
   pep8-1.5.3-py2.py3-none-any.whl#md5=0654904760aa9a24062bf367f39e873e)]
```
Thanks!

- Any questions?
Thanks!

- Any questions?
- Bon appétit!