Pipeline Documentation

Release 1.1.27

Timothée Peignier

February 02, 2012

CONTENTS

Pipeline is an asset packaging library for Django, providing both CSS and JavaScript concatenation and compression, built-in JavaScript template support, and optional data-URI image and font embedding.

You can report bugs and discuss features on the issues page.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

1. Either check out Pipeline from GitHub or to pull a release off PyPI

   ```
   pip install django-pipeline
   ```

2. Add 'compress' to your `INSTALLED_APPS`

   ```
   INSTALLED_APPS = (
       'pipeline',
   )
   ```

1.1.1 Recommendations

By default Pipeline uses YUI Compressor to compress CSS and JS. YUI Compressor is an excellent stand-alone application for dealing with JS and CSS-files. YUI Compressor can be downloaded from: http://developer.yahoo.com/yui/compressor/.

If you do not install YUI Compressor, make sure to disable the compressor in your settings.

1.2 Configuration

Configuration and list of available settings for Pipeline

Note: If you are updating from django-compress or from previous versions of django-pipeline, don't forget to read *Changelog*.

1.2.1 Specifying files

You specify groups of files to be compressed in your settings. You can use glob syntax to select multiples files.

The basic syntax for specifying CSS/JavaScript groups files is

```
PIPELINE_CSS = {
    'group_one': {
        'source_filenames': (
          'css/style.css',
```

```
            'css/foo.css',
            'css/button/*.css',
            'css/bar.css'
        ),
        'output_filename': 'css/one_compressed.css',
        'extra_context': {
            'media': 'screen,projection',
        },
    },
    # other CSS groups goes here
}

PIPELINE_JS = {
    'all': {
        'source_filenames': (
          'js/jquery-1.2.3.js',
          'js/jquery-preload.js',
          'js/jquery.pngFix.js',
          'js/my_script.js',
          'js/my_other_script.js'
        ),
        'output_filename': 'js/all_compressed.js',
    }
}
```

Group options

`source_filenames`

Required

Is a tuple with the source files to be compressed. The files are concatenated in the order it is specified in the tuple.

`output_filename`

Required

Is the filename of the (to be) compressed file.

`variant`

Optional

Is the variant you want to apply to your CSS. This allow you to embed images and fonts in CSS with data-URI or MHTML. Allowed values are : `None`, `datauri` or `mhtml`.

Defaults to `None`.

`manifest`

Optional

Indicate if you want this group to appear in your cache manifest.

Defaults to `True`.

`extra_context`

Optional

Is a dictionary of values to add to the template context, when generating the HTML for the HTML-tags with the templatetags.

For CSS, if you do not specify `extra_context/media`, the default media in the `<link>` output will be `media="all"`.

`absolute_asset_paths`

Optional

Indicates if relative paths in CSS files should be made absolute, based on `PIPELINE_URL`. This only applies to entries in `PIPELINE_CSS`.

Defaults to `True`.

Note: Note that all filenames are specified relative to `PIPELINE_ROOT`, and thus the source files needs to be in your `PIPELINE_ROOT`.

Other settings

`PIPELINE`

When `PIPELINE` is `True`, CSS and JavaScripts will be concatenated and filtered. When `False`, the source-files will be used instead.

Defaults to `not DEBUG` (compressed files will only be used when `DEBUG` is `False`).

`PIPELINE_AUTO`

Auto-generate CSS and JavaScript files whenever needed, when the template tags are invoked.

This setting will make sure that the outputted files always are up to date (assuming that you are using the provided templatetags to output the links to your files).

If you disable this, you can use the management command to keep your files manually updated.

Defaults to `True`.

`PIPELINE_VERSION`

Regulates whether or not to add a “version number” to the outputted files filename with for use with “far future Expires”.

When you specify `PIPELINE_VERSION` you will also need to add a placeholder (which by default is `?`) for the version number in the `output_filename` setting.

PIPELINE_VERSION_REMOVE_OLD

When `True`, old compressed files will be removed when new versions are generated. All files with a matching name e.g. `output_filename` where `?` can be replaced by digits will be removed.

If you for some reason have files named in the same way, you should consider moving them or putting the compressed files in their own directory.

Defaults to `True`.

Example:

```
PIPELINE = True
PIPELINE_VERSION = True
PIPELINE_CSS = {
    'screen': {
        'source_filenames': (
            'css/screen/style.css', 'css/screen/paginator.css',
            'css/screen/agenda.css', 'css/screen/weather.css',
            'css/screen/gallery.css',
        ),
        'output_filename': 'c/screen.r?.css',
    },
}
```

This will output a file like `/media/c/screen.r1213947531.css`, which will be re-generated and updated when you change your source files.

PIPELINE_CSS_COMPRESSOR

Compressor class to be applied to CSS files.

If empty or `None`, CSS files won't be compressed.

Defaults to `'pipeline.compressors.yui.YUICompressor'`.

PIPELINE_JS_COMPRESSOR

Compressor class to be applied to JavaScript files.

If empty or `None`, JavaScript files won't be compressed.

Defaults to `'pipeline.compressors.yui.YUICompressor'`

Note: Please note that in order to use YUI Compressor, you need to install YUI Compressor (see *Installation* for more details).

PIPELINE_TEMPLATE_NAMESPACE

Object name where all of your compiled templates will be added, from within your browser. To access them with your own JavaScript namespace, change it to the object of your choice.

Defaults to `"window.JST"`

`PIPELINE_TEMPLATE_EXT`

The extension for which Pipeline will consider the file as a Javascript templates. To use a different extension, like `.mustache`, set this settings to `.mustache`.

Defaults to `".jst"`

`PIPELINE_TEMPLATE_FUNC`

JavaScript function that compiles your JavaScript templates. Pipeline doesn't bundle a javascript template library, but the default settings is to use the underscore template function.

Defaults to `"_.template"`

`PIPELINE_CACHE_TIMEOUT`

Package version are cached to avoid unnecessary IO, the default is to cache version for 2 years.

Defaults to `63072000`

1.2.2 Embedding fonts and images

You can embed fonts and images directly in your compiled css, using Data-URI in modern browser or MHTML in Internet Explorer 7 or below.

To do so, setup variant group options to the method you wish to use :

```
PIPELINE_CSS = {
    'master': {
        'source_filenames': (
          'css/core.css',
          'css/button/*.css',
        ),
        'output_filename': 'css/master.css',
        'variant': 'datauri',
    },
}
```

Images and fonts are embedded following these rules :

- If asset is under **32 kilobytes** to avoid rendering delay or not rendering at all in Internet Explorer 8.
- If asset path contains a directory named "**embed**".

1.2.3 Rewriting CSS urls

If source CSS contain a relative URL (i.e. relative to current file), and `absolute_asset_paths` is set to `True` or left out in the package entry, the URL will be converted to full relative path using `PIPELINE_URL`. This conversion is performed before any compressors are applied

```
media/js/fancybox/
  fancybox.png
  fancybox-x.png
  fancybox-y.png
  jquery.fancybox-1.3.4.css
  jquery.fancybox-1.3.4.js
```

jquery.fancybox-1.3.4.css contains

```
background-image: url('fancybox.png');
background-image: url('fancybox-x.png');
background-image: url('fancybox-y.png');
```

In resulting CSS it will be rewritten to

```
background-image:url(/js/fancybox/fancybox.png);
background-image:url(/js/fancybox/fancybox-x.png);
background-image:url(/js/fancybox/fancybox-y.png);
```

(Assuming `PIPELINE_URL` is '' or '/', with non-empty `PIPELINE_URL` result will be another).

1.2.4 External urls

Warning: This feature is currently deprecated and will be remove in next major version of pipeline.

While Pipeline does a great job of minimizing the amount of http requests on your site (hence increasing performance) there are sometimes cases when you want to include external files as well. Let's take an example:

```
PIPELINE_JS = {
    'jquery': {
        'external_urls': (
            'http://ajax.googleapis.com/ajax/libs/jquery/1.2.6/jquery.min.js',
            'http://ajax.googleapis.com/ajax/libs/jqueryui/1.5.2/jquery-ui.min.js'
        ),
    },
    'all': {
        'source_filenames': ('js/blog.js', 'js/comments.js'),
        'output_filename': 'js/all.js',
    },
}
```

In template:

```
{% load compressed %}
{% compressed_js 'jquery' %}
{% compressed_js 'all' %}
```

Output in when `settings.PIPELINE = False`:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.2.6/jquery.min.js"
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.5.2/jquery-ui.mi
<script type="text/javascript" src="/media/js/blog.js" charset="utf-8"></script>
<script type="text/javascript" src="/media/js/comments.js" charset="utf-8"></script>
```

Output in when `settings.PIPELINE = True`:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.2.6/jquery.min.js"
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.5.2/jquery-ui.mi
<script type="text/javascript" src="/media/js/all.js" charset="utf-8"></script>
```

Now why is this good you ask? The more script sources the more impact on performance according to http://developer.yahoo.com/performance/rules.html#num_http which is true but if you are low bandwidth or super-big you may want to offload some horsepower to google which leads us as hinted in the example above to the next topic.

Note: External urls is currently only available for javascript.

1.3 Usage

Describes how to use Pipeline when it is installed and configured.

1.3.1 Automated generation

If `PIPELINE` and `PIPELINE_AUTO` is enabled (`True`), the source files will be automatically updated, and re-generated if needed when invoked from the templatetags. The last modified time of the files will be compared, and if any of the source-files is newer than the output-file, the file will be re-generated.

1.3.2 Management command

You can update and force updates of the compressed file(s) with the management command "synccompress". This makes it possible to keep the files updated manually.

The command is (assuming you are in you project-folder that contains `manage.py`)

```
./manage.py synccompress
```

To force all files to be re-generated, use the argument `--force`

```
./manage.py synccompress --force
```

To re-generate only a specific group

```
./manage.py synccompress screen
```

To re-generate only specific groups

```
./manage.py synccompress screen print
```

1.3.3 Templatetags

Pipeline includes two template tags: `compressed_css` and `compressed_js`, in a template library called `compressed`.

They are used to output the `<link>` and `<script>`-tags for the specified CSS/JavaScript-groups (as specified in the settings). The first argument must be the name of the CSS/JavaScript group.

The templatetags will either output the source filenames or the compressed filenames, depending on the `PIPELINE` setting, if you do not specify the `PIPELINE` setting, the source files will be used in DEBUG-mode, and compressed files in non-DEBUG-mode.

If you need to change the output of the HTML-tags generated from the templatetags, this can be done by overriding the templates `pipeline/css.html` and `pipeline/js.html`.

Example

If you have specified the CSS-groups "screen" and "print" and a JavaScript-group with the name "scripts", you would use the following code to output them all

```
{% load compressed %}
{% compressed_css 'screen' %}
{% compressed_css 'print' %}
{% compressed_js 'scripts' %}
```

1.3.4 Middleware

To enable HTML compression add `pipeline.middleware.MinifyHTMLMiddleware`, to your `MIDDLEWARE_CLASSES` settings.

Ensure that it comes after any middleware which modify your HTML, like `GZipMiddleware`

```
MIDDLEWARE_CLASSES = (
   'django.middleware.gzip.GZipMiddleware',
   'pipeline.middleware.MinifyHTMLMiddleware',
)
```

1.3.5 Cache manifest

Pipeline provide a way to add your javascripts and stylesheets files to a cache-manifest via Manifesto.

To do so, you just need to add manifesto app to your `INSTALLED_APPS`.

1.4 Compressors

1.4.1 YUI Compressor compressor

The YUI compressor use yui-compressor for compressing javascript and stylesheets.

To use it for your stylesheets add this to your `PIPELINE_CSS_COMPRESSOR`

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.yui.YUICompressor'
```

To use it for your javascripts add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.yui.YUICompressor'
```

`PIPELINE_YUI_BINARY`

Command line to execute for the YUI program. You will most likely change this to the location of yui-compressor on your system.

Defaults to `'/usr/local/bin/yuicompressor'`.

Warning: Don't point to `yuicompressor.jar` directly, we expect to find a executable script.

PIPELINE_YUI_CSS_ARGUMENTS

Additional arguments to use when compressing CSS.

Defaults to ''.

PIPELINE_YUI_JS_ARGUMENTS

Additional arguments to use when compressing JavaScript.

Defaults to ''.

1.4.2 Closure Compiler compressor

The Closure compressor use Google Closure Compiler to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = (
  'pipeline.compressors.closure.ClosureCompressor',
)
```

PIPELINE_CLOSURE_BINARY

Command line to execute for the Closure Compiler program. You will most likely change this to the location of closure on your system.

Default to `'/usr/local/bin/closure'`

Warning: Don't point to `compiler.jar` directly, we expect to find a executable script.

PIPELINE_CLOSURE_ARGUMENTS

Additional arguments to use when closure is called.

Default to ''

1.4.3 UglifyJS compressor

The UglifyJS compressor use UglifyJS to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.uglifyjs.UglifyJSCompressor'
```

PIPELINE_UGLIFYJS_BINARY

Command line to execute for the Closure Compiler program. You will most likely change this to the location of closure on your system.

Defaults to `'/usr/local/bin/uglifyjs'`.

PIPELINE_UGLIFYJS_ARGUMENTS

Additional arguments to use when uglifyjs is called.

Default to ''

1.4.4 JSMin compressor

The jsmin compressor use Douglas Crockford jsmin tool to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.jsmin.JSMinCompressor'
```

1.4.5 CSSTidy compressor

The CSStidy compressor use CSStidy to compress stylesheets.

To us it for your stylesheets add this to your `PIPELINE_CSS_COMPRESSOR`

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.csstidy.CSSTidyCompressor'
```

PIPELINE_CSSTIDY_BINARY

Command line to execute for csstidy program. You will most likely change this to the location of csstidy on your system.

Defaults to `'/usr/local/bin/csstidy'`

PIPELINE_CSSTIDY_ARGUMENTS

Additional arguments to use when csstidy is called.

Default to `'--template=highest'`

1.4.6 Write your own compressor class

To write your own compressor class, for example want to implement other types of compressors.

All you need to do is to create a class that inherits from `pipeline.compressors.CompressorBase` and implements `compress_css` and/or a `compress_js` when needed.

Finally, add it to `PIPELINE_CSS_COMPRESSOR` or `PIPELINE_JS_COMPRESSOR` settings (see *Configuration* for more information).

Example

A custom compressor for a imaginary compressor called jam

```python
from pipeline.compressors import CompressorBase

class JamCompressor(CompressorBase):
  def compress_js(self, js):
    return jam.compress(js)

  def compress(self, css):
    return jam.compress(css)
```

Add it to your settings

```
PIPELINE_CSS_COMPRESSOR = 'jam.compressors.JamCompressor'
PIPELINE_JS_COMPRESSOR = 'jam.compressors.JamCompressor'
```

1.5 Compilers

1.5.1 Coffee Script compiler

The Coffee Script compiler use Coffee Script to compile your javascripts.

To use it add this to your `PIPELINE_COMPILERS`

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.coffee.CoffeeScriptCompiler',
)
```

PIPELINE_COFFEE_SCRIPT_BINARY

Command line to execute for coffee program. You will most likely change this to the location of coffee on your system.

Defaults to `'/usr/local/bin/coffee'`.

PIPELINE_COFFEE_SCRIPT_ARGUMENTS

Additional arguments to use when coffee is called.

Defaults to ''.

1.5.2 LESS compiler

The LESS compiler use LESS to compile your stylesheets.

To use it add this to your `PIPELINE_COMPILERS`

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.less.LessCompiler',
)
```

PIPELINE_LESS_BINARY

Command line to execute for lessc program. You will most likely change this to the location of lessc on your system.

Defaults to `'/usr/local/bin/lessc'`.

PIPELINE_LESS_ARGUMENTS

Additional arguments to use when lessc is called.

Defaults to ''.

1.5.3 SASS compiler

The SASS compiler use SASS to compile your stylesheets.

To use it add this to your `PIPELINE_COMPILERS`

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.sass.SASSCompiler',
)
```

PIPELINE_SASS_BINARY

Command line to execute for sass program. You will most likely change this to the location of sass on your system.

Defaults to `'/usr/local/bin/sass'`.

PIPELINE_SASS_ARGUMENTS

Additional arguments to use when sass is called.

Defaults to ''.

1.5.4 Stylus compiler

The Stylus compiler use *Stylus <http://learnboost.github.com/stylus/>* to compile your stylesheets.

To use it add this to your `PIPELINE_COMPILERS`

```
PIPELINE_COMPILERS = (
    'pipeline.compilers.stylus.StylusCompiler',
)
```

PIPELINE_STYLUS_BINARY

Command line to execute for stylus program. You will most likely change this to the location of stylus on your system.

Defaults to `'/usr/local/bin/stylus'`.

PIPELINE_STYLUS_ARGUMENTS

Additional arguments to use when stylus is called.

Defaults to ''.

1.5.5 Write your own compiler class

To write your own compiler class, for example want to implement other types of compilers.

All you need to do is to create a class that inherits from `pipeline.compilers.CompilerBase` and implements `match_file` and `compile_file` when needed.

Finally, specify it in the tuple of compilers `PIPELINE_COMPILERS` in the settings.

Example

A custom compiler for a imaginary compiler called jam

```
from pipeline.compilers import CompilerBase

class JamCompiler(CompilerBase):
  output_extension = 'js'

  def match_file(self, filename):
    return path.endswith('.jam')

  def compile_file(self, content, path):
    return jam.compile(content)
```

1.6 Versioning

Warning: This feature is currently deprecated and will be remove in next major version of pipeline.

There are several ways for generating version strings. Basically, three types are available. These are:

- mtime
- hash
- git

1.6.1 Modification time version

This is the default method for generating version strings. In short, when invoked, it checks whether any of the source files system timestamps (mtime) is newer than the version string of the corresponding compressed file. If that is the case, the compressed output file version string will be the mtime of the most recent source file.

1.6.2 Hash version

Hash-based versioning works by generating a hash string based on the contents of the source files. Available hash-based versioning methods are MD5 and SHA-1.

MD5 version

To generate MD5 version strings, put this in your *settings.py*

```
PIPELINE_VERSIONING = 'pipeline.versioning.hash.MD5Versioning'
```

SHA-1 version

To generate SHA-1 version strings, put this in your *settings.py*:

```
PIPELINE_VERSIONING = 'pipeline.versioning.hash.SHA1Versioning'
```

1.6.3 Git version

Versions formed on git revisions in codebase. Provides a fast way to check if any of your files changed that will be consistent across multiple web servers so that they all generate the same version numbers for each set of source files, assuming their git repositories are all in sync.

Assumes deployment is git repositiory. Requires GitPython 0.2.0. GitPython 0.3.0 uses an async library that does not currently play well with Django. To install using Git just do pip install GitPython==0.2.0-beta1.

Git revision version

To generate versions based on revision of every file in your source file list, put this in your *settings.py*:

```
PIPELINE_VERSIONING = 'pipeline.versioning.git.GitVersioningBase'
```

Git HEAD last revision version

To generate versions based on the latest revision of HEAD in your git repository (which assumes all of your source files are in the same repository), put this in your *settings.py*:

```
PIPELINE_VERSIONING = 'pipeline.versioning.git.GitHeadRevVersioning'
```

1.6.4 Write your own versioning class

To write your own versioning class, you can subclass one of the available versioning classes.

Example

For example, you want a short version string based on the SHA-1 version string. You can do this by subclassing the SHA1Versioning class and overriding its get_version() method, like this:

```python
from pipeline.versioning.hash import SHA1Versioning

class ShortSHA1Versioning(SHA1Versioning):
    """Custom SHA1Versioning class"""

    def get_version(self, source_files):
        """Return the first 10 characters of the SHA-1 version string"""
        version = super(ShortSHA1Versioning, self).get_version(source_files)
        return version[:10]
```

In your `settings.py` add:

```
PIPELINE_VERSIONING = 'app.module.ShortSHA1Versioning'
```

Note: Replace `app` and `module` by the app and module that contains your versioning class

1.6.5 Production environment

You probably do not want the source files to be evaluated and (if needed) regenerated on every request in a production environment. Especially when calculating a hash on every request could be expensive. To avoid this, make sure your source files are compressed before deployment, and put the following settings in your production environment's `settings.py`:

```
PIPELINE_AUTO = False
PIPELINE_VERSION = True
```

This way, the names of the compressed files will be looked up in the file system, instead of being evaluated and (if needed) regenerated on every request.

1.7 Javascript Templates

Pipeline allows you to use javascript templates along with your javascript views. To use your javascript templates, just add them to your `COMPRESS_JS` group

```
COMPRESS_JS = {
  'application': {
    'source_filenames': (
      'js/application.js',
      'js/templates/**/*.jst',
    ),
    'output_filename': 'js/application.r?.js'
  }
}
```

For example, if you have the following template `js/templates/photo/detail.jst`

```
<div class="photo">
 <img src="<%= src %>" />
 <div class="caption">
  <%= caption %>
 </div>
</div>
```

They will be available from your javascript code via window.JST

```
JST.photo_detail({ src:"images/baby-panda.jpg", caption:"A baby panda is born" });
```

1.7.1 Configuration

Template function

By default, it use underscore template function, but without providing it. You can specify your own template function via `PIPELINE_TEMPLATE_FUNC`

```
PIPELINE_TEMPLATE_FUNC = 'new Template'
```

Template namespace

Your templates are made available in a top-level object, by default `window.JST`, but you can choose your own via `PIPELINE_TEMPLATE_NAMESPACE`

```
PIPELINE_TEMPLATE_NAMESPACE = 'window.Template'
```

Template extension

Templates are detected by their extension, by default `.jst`, but you can use your own extension via `PIPELINE_TEMPLATE_EXT`

```
PIPELINE_TEMPLATE_EXT = '.mustache'
```

1.7.2 Using it with your favorite template library

Mustache

To use it with Mustache you will need this some extra javascript

```
Mustache.template = function(templateString) {
  return function() {
    if (arguments.length < 1) {
      return templateString;
    } else {
      return Mustache.to_html(templateString, arguments[0], arguments[1]);
    }
  };
};
```

And use this settings

```
PIPELINE_TEMPLATE_EXT = '.mustache'
PIPELINE_TEMPLATE_FUNC = 'Mustache.template'
```

Prototype

To use it with Prototype, just setup your `PIPELINE_TEMPLATE_FUNC`

```
PIPELINE_TEMPLATE_FUNC = 'new Template'
```

1.8 Storages

1.8.1 Using with a custom storage

Pipeline use Django Storage to read, save and delete files, by default it use an improved `FileSystemStorage`.

You can provide your own via `PIPELINE_STORAGE` :

```
PIPELINE_STORAGE = 'storages.backends.s3boto.S3BotoStorage'
```

1.8.2 Using with staticfiles

Pipeline is providing a Finder for staticfiles app, to use it configure `STATICFILES_FINDERS` like so :

```
STATICFILES_FINDERS = (
  'pipeline.finders.PipelineFinder',
  'django.contrib.staticfiles.finders.FileSystemFinder',
  'django.contrib.staticfiles.finders.AppDirectoriesFinder'
)
```

Note: `PipelineFinder` should be the first finder in `STATICFILES_FINDERS`.

Pipeline is also providing a storage that play nicely with staticfiles app particularly for development :

```
PIPELINE_STORAGE = 'pipeline.storage.PipelineFinderStorage'
```

1.9 Signals

A list of all signals send by pipeline.

1.9.1 css_compressed

pipeline.signals.css_compressed

Whenever a css package is compressed, this signal is sent after the compression.

Arguments sent with this signal :

sender The `Packager` class that compressed the group.

package The package actually compressed.

version The version identifier if the newly compressed file.

1.9.2 js_compressed

pipeline.signals.js_compressed

Whenever a js package is compressed, this signal is sent after the compression.

Arguments sent with this signal :

sender The `Packager` class that compressed the group.

package The package actually compressed.

version The version identifier if the newly compressed file.

1.10 Sites using Pipeline

The following sites are a partial list of people using Pipeline.

Are you using pipeline and not being in this list ? Drop us a line.

1.10.1 20 Minutes

For their internal tools : http://www.20minutes.fr

1.10.2 Pitchfork

For their main website : http://pitchfork.com

1.10.3 The Molly Project

Molly is a framework for the rapid development of information and service portals targeted at mobile internet devices : http://mollyproject.org

It powers the University of Oxford's mobile portal : http://m.ox.ac.uk/

1.10.4 Croisé dans le Métro

For their main and mobile website :

- http://www.croisedanslemetro.com
- http://m.croisedanslemetro.com

1.11 Changelog

1.11.1 1.1.27

- Improve windows support.
- Add support for Django 1.2. Thanks to Balazs Kossovics.

1.11.2 1.1.26

- Fix unicode issue. Thanks to Adam Charnock.

1.11.3 1.1.25

- Add stylus compiler. Thanks to David Charbonnier.
- Fix git versioning. Thanks to David Charbonnier again.
- Allow to disable asset normalization. Thansk to Christian Hammond.

1.11.4 1.1.24

- Add deprecation warning on external urls
- Fix windows paths support.
- Don't try to pack externals package. Thanks to Cristian Andreica for the report.

1.11.5 1.1.23

- Improve compressors documentation.
- Allow to have no compressors. Thanks to Christian Hammond.
- Fix less compiler to support @import. Thanks to TK Kocheran.

1.11.6 1.1.22

- Fix google closure compiler verbose mode. Thanks to cgreene.
- Fix absolute_path generation. Thanks to cgreene and Chris Northwood.

1.11.7 1.1.21

- Fix js template name generation when there is only one template. Thanks to Gerrdo Curiel for the report.

1.11.8 1.1.20

- Fix less and sass compilers.
- Properly overwrite compiled file if it already exists.

1.11.9 1.1.19

- Add python 2.5 support.
- Remove lessc default arguments.

1.11.10 1.1.18.1

- Ensure we don't break for font-face hack and other esoteric CSS urls.

1.11.11 1.1.18

- Ensure we don't break font-face urls.

1.11.12 1.1.17

- Don't use verbose where it's not supported or source of error.
- Improve syncompress cache bsuting ability.

1.11.13 1.1.16

- Add a way to compress specific groups with `synccompress` command.

1.11.14 1.1.15.2

- Fixing typo.

1.11.15 1.1.15

- Allow synccompress to only update version cache.

1.11.16 1.1.14.1

- Fix bug when calculating relative_path.

1.11.17 1.1.14

- Ensure javascript templates code get compiled properly.
- Use os.path.relpath() instead of the more error-prone string.replace() to find the relative path. Thanks to Luke Yu-Po Chen.

1.11.18 1.1.13

- Use `storage.save` in `Packager`, to play nicely with exotic storage.

1.11.19 1.1.12

- Add configurable cache timeout. Thanks to Matt Dennewitz.
- Catch any exception coming from `storage`.

1.11.20 1.1.11

- Add documentation on pipeline signals, see *Signals*.
- Cache version calculations, speeding up template tags.
- Not assuming anymore that version identifier are sortable (beware if you have setup `PIPELINE_VERSION_REMOVE_OLD` to `False`).

1.11.21 1.1.10

- Make `synccompress` command work as expect when `PIPELINE_AUTO` is set to `False`.
- Add a way to compress a specific group with `synccompress` command.
- Raise `CompilerError` when `PIPELINE` is set to `False`.

1.11.22 1.1.9

- Play more nicely with staticfiles app via `PipelineFinderStorage`, see *Storages*.

1.11.23 1.1.8.1

- Fix bug in asset absolute path rewriting.

1.11.24 1.1.8

- Faster templates tags.
- Storage speed up.

1.11.25 1.1.7

- Improved windows support. Thanks to Kyle MacFarlane.
- Added Manifesto support.

1.11.26 1.1.0

- Most of the settings name have change to be prefixed by `PIPELINE_`.
- CSSTidy isn't the default anymore, YUI Compressor is now the default.
- Filters are now called compressors.
- You can only specify one compressor via `PIPELINE_CSS_COMPRESSOR` or `PIPELINE_JS_COMPRESSOR`

CHAPTER

TWO

INDICES AND TABLES

- *search*