Richard Documentation

Release 0.2.dev

Will Kahn-Greene

May 21, 2015

Contents

richard is a video indexing web application. It makes it easy to set up a website that indexes video that's hosted on other sites like YouTube, Vimeo, blip.tv, etc.

CHAPTER 1

Part 1: Site Administrator's Guide

This guide is for site administrators and covers everything you need to know to run a richard-based video index site.

1.1 What's new in richard

1.1.1 Version 0.2: In development

Features:

- New API v2! Built on django-rest-framework, it should be simpler, easier, and less buggy. See documentation for API details.
- Switched to django-configurations. You'll have to update your settings files, but understanding where settings are coming from should now be much easier.

Backwards-incompatible changes:

- Ditched API v1. It used tastypie and I jettisoned that.
- To use API v2, you'll need to create new API tokens in the admin in the Token section.
- Switched from Jinja2 templates to Django templates. If you had your own theme, you'll have to redo it with Django templates.

Other changes:

- Moved a bunch of files from `richard/` to the new base app in `richard/base/`. This cleans up the richard project directory a bit.
- Updated to django-haystack 2.0.
- Cleaned up the search page a bit. Added a category facet.
- Cleaned up video listing pages.
- Category pages now show draft videos if you're an admin.
- Updated to Django 1.6.
- Updated requirements.

1.1.2 Version 0.1: May 18th, 2013

Features:

- categories, speakers, videos
- lots of metadata
- RSS feeds for videos, categories, site news
- site news
- basic search
- admin section
- JSON API for querying the data and adding data

Backwards-incompatible changes:

- None since this is the initial release.

Other changes:

- Everything since this is the intitial release.

Thank you!:

Thank you to all the following folks who contributed code or reported bugs or otherwise helped with this release!

- Adam Zapletal
- Alejandro Gómez
- Andrew Szeto
- Carl Karsten
- Chris Lawlor
- David Winterbottom
- Eduard-Cristian Stefan
- Eka Putra
- Fernando Gutierrez
- Frank Ploss
- Gledi Caushaj
- lukegotszling
- Miki Tebeka
- Nathan Villaescusa
- Nes.quik
- Pavel Savchenko
- Reiner Gerecke
- Ricky Rosario
- Riley Labrecque
- Roy Hyunjin Han
- Senthil Kumaran

- Sheila Miguez
- Spencer Herzberg
- Will Kahn-Greene

1.2 Installing richard

FIXME: This needs to be written. For now, you can look at the *Installing, running and testing richard*, use postgresql and skip the things that are development-environment specific.

1.2.1 Settings

richard relies extensively on environment settings which **will not work with Apache/mod_wsgi setups**.

For configuration purposes, the following table maps the 'richard' environment variables to their Django setting:

Environment Variable	Django Setting	Development Default	Production Default
DJANGO_DATABASES	DATABASES	See code	See code
DJANGO_DEBUG	DEBUG	True	False
DJANGO_SECRET_KEY	SECRET_KEY	secret-value	raises error
RICHARD_API	API	True	False
RICHARD_SITE_URL	SITE_URL	http://127.0.0.1:8000	http://127.0.0.1:8000

1.3 Upgrading

1.3.1 Upgrading 0.1 to 0.2

1. Uninstall South in your virtual environments.
2. Run `./manage.py migrate --fake` to mark all the migrations as having run because we switched from Django 1.6 with South to Django 1.7 migration systems.

1.3.2 Up to 0.1

Note: When you upgrade to a new version of richard, please please please read through the What's New notes for all the versions between what you're running and what you're upgrading to **BEFORE** you start your upgrade process.

The general list of things you need to do when you upgrade to a new version is as follows:

1. backup your database.
2. backup your virtual environment and code.

Basically, you want to be able to go back to square 1 if something goes awry. If you don't do this effectively and something does go awry, you'll feel very irritated with yourself.

After doing that, you should:

1. upgrade the richard software
2. upgrade your virtual environment:

```
$ pip install .
```

3. update your settings files—details on what needs to be updated will be in the What's New notes
4. upgrade your database:

```
$ ./manage.py migrate
```

5. upgrade your index:

```
$ ./manage.py rebuild_index
```

6. upgrade your templates—details on what needs to be updated will be in the What's New notes

1.4 Making it faster

1.4.1 django-staticgenerator

If you're using richard in a predominantly read-only way, then you should take a look at using django-staticgenerator. It takes pages generated and stores them on disk. Then you can have your web-server serve the file on disk instead of kicking off the django application which is way faster.

I'm using Apache with mod_wsgi, so I used the instructions here.

1.5 Running a richard site

Note: The information in this chapter is likely to change as we spend more time on the admin side of things.

1.5.1 Changing the theme/style/look-n-feel

Override the CSS, JS and templates.

Todo

flesh out how to override css, js and templates

1.5.2 Using the admin tool

1. Enter the admin are of your site. This is often the url for your site plus `/admin/`.

   e.g. For a richard installation at http://example.com/, the admin would be at http://example.com/admin/.

   Note: If you want a link to the admin page from the fron page of your site, just modify your site templates.

2. If you haven't logged into the admin, you will need to do so now. Use the superuser account you created when installing richard.

Once in the admin tool, you have access to all the data on the site including site news, site-wide notifications, videos, categories, tags, and speakers.

1.5.3 Adding videos

1. Enter the admin tool.
2. Once logged in, click on *Videos* on the left side.
3. Click on *Add Video* in the upper right hand side.
4. Fill out the information to add the video.

Here are a couple of things to know that might help you:

1. Keep it in DRAFT mode until you're done with it. This keeps it from showing up in any of the video lists, but you can still see the video if you go directly to its page.
2. Make sure to *save and continue editing* from time to time. After doing that, you can click on *View on site* in the upper right hand corner and see what the video looks like on the site.
3. The summary and description fields are in HTML. Use structure and semantic tags like <p>, <ul>, <ol>, <li>, <a>, ...

   Don't use formatting tags. Otherwise when you decide to change the style of your site, you'll have to go through and change all the summaries and descriptions, too.

   Example:

```
<p>
  This video covers how to use nose for unit testing. It follows
  this outline:
</p>
<ul>
  <li>test discovery
  <li>writing tests
  <li>nose helper tools
</ul>
...
```

1.5.4 Editing videos

1. Enter the admin tool.
2. Once logged in, click on *Videos* on the left side.
3. Find the video you want to edit and then click on the video to edit it.

Make sure to *save and continue editing* from time to time. After doing that, you can click on *View on site* in the upper right hand corner and see what the video looks like on the site.

Note: If you're already logged in as an admin, every video page on the site will have an *edit* link at the top of the metadata. If you click that, it'll bring you directly to the video edit page in the admin.

1.5.5 Removing videos

1. Enter the admin tool.
2. Once logged in, click on *Videos* on the left side.
3. Find the video you want to delete and then click on the video to go to the edit screen.
4. Click on the *delete* link in the top left.

Warning: Deleting videos is currently permanent! There's no way to undo this action.

1.5.6 Importing a batch of videos

There is currently no way to import a batch of videos all at once using Richard.

There are some non-Richard options:

1. You could write a script that runs on your server that accesses the model objects directly. That's what I've been doing so far. That requires Python/Django chops.
2. You could enable the API and write code that uses the API to add and update videos in Richard.
3. You could use steve which is the tool I've been working on which uses the API to add and update videos. It's very prototype-quality. If you're interested in helping out, please do.

1.5.7 Updating the search index

Updating the index is a manual process. It's possible we could update the index as data is changed, but richard is currently not set up that way.

Do this to update the index:

```
./manage.py rebuild_index
```

You probably want to either put this in a cron job or run it after making any data changes.

Warning: This deletes the index, then rebuilds it. Thus there will be a period of time during which search on your site will kind of suck.

1.5.8 Adding static pages

The "About" page is a static page. Our system lets you add pages as you so desire. To add a page:

1. Create the page as a Django template file in `templates/pages/<page-name>`.

   For example, if I wanted to create a page for contact information, I'd create `templates/pages/contact.html` which would have in it:

```
{% extends "base.html" %}
{% block title %}{% page_title 'About' %}{% endblock %}
{% block content %}

<div class="page-header">
  <h1>Contact information</h1>
</div>

<div class="row">
  <div class="span12">
    <p>
      If you have problems with this site, send email to
      joe@example.com.
    </p>
  </div>
</div>
{% endblock %}
```

2. Add that page to `PAGES` in your settings file.

   Using the above example, we'd change `PAGES` to:

```
PAGEs = ['about', 'contact']
```

   Now our contact page is available at the url `http://example.com/pages/contact`.

1.6 API

richard comes with an REST API built with django-rest-framework.

1.6.1 Enabling the API

The API is disabled by default. To enable the API, add this to your *richard/settings_local.py* file:

```
API = True
```

1.6.2 API tokens and authenticating

Note: The API keys used in richard v 0.1 are different than the API tokens used in richard v 0.2 and later. If you were using richard v0.1, you'll need to create new API tokens when you upgrade to v0.2.

Anonymous users have:

- read-only access to video data

Authenticated users have:

- read/create/update access to video data

To authenticate, you need a valid API token. To get one, you need to:

1. log into the richard admin
2. click on *Tokens* in the *Authtoken* section
3. click on *Add token* in the upper right
4. select the user you want to add a key for in the drop down
5. click on *Save* in bottom right hand corner

After doing that, your API token will be generated and will be in the *Key* field.

Use the *Authorization* HTTP header to authenticate. The value is your API token.

For example, using curl:

```
$ curl --dump-header - \
    -H "Content-Type: application/json" \
    -H "Authorization: Token abd984049d938e8909ff" \
    -X POST 'http://example.com/api/v2/video/' \
    -d '
{
    "tag": "foo"
    ...
}'
```

Obviously, that data doesn't work, but the header structure is correct.

1.6.3 API

Category

GET /api/v2/category/ Lists categories.

Example:

```
$ curl -X GET 'http://example.com/api/v2/category/'
```

GET /api/v2/category/<CATEGORY_SLUG>/ Lists information about that category.

Example:

```
$ curl -X GET 'http://example.com/api/v2/category/pycon-2011/'
```

Speaker

GET /api/v2/speaker/ Lists speakers.

Example:

```
$ curl -X GET 'http://example.com/api/v2/speaker/'
```

Videos

GET /api/v2/video/ Lists all the videos on the site. This is paginated. The pagination fields are at the top level and titled *next* and *previous*.

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/'
```

GET /api/v2/video/<VIDEO_ID>/ Returns information for that specific video id.

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/2/'
```

GET /api/v2/video/?speaker=FOO Returns videos with speaker FOO. It only handles one speaker, but it uses *icontains* on the speaker field which will do case-insensitive substring matches.

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/?speaker=asheesh'
```

GET /api/v2/video/?tag=FOO Returns videos with tag FOO. It only takes one tag and does an exact match.

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/?tag=django'
```

GET /api/v2/video/?category=FOO Returns videos with category FOO. It only takes one category and does an exact match on the category slug.

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/?category=pycon-us-2014'
```

GET /api/v2/video/?ordering=FOO Returns videos ordered by field FOO. The fields allowed in the ordering filter are *added*, *recorded* and *title*. Reverse ordering is specified by prefixing the field name with -

Example:

```
$ curl -X GET 'http://example.com/api/v2/video/?ordering=-added'
```

POST /api/v2/video/ Creates a new video.

PUT /api/v2/video/<VIDEO_ID>/ Updates an existing video.

Note: You can only update videos in DRAFT mode. If it's live, you will get an error.

Fields for creating/updating videos:

category — Required. The title of the category.

The category must exist on the site. If it doesn't exist, the API will waggle its finger at you. (Oops!)

Example: `"category": "PyCon 2012"`

title — Required. The title of the video.

Example: `"title": "My dog has fleas"`

language — Required. Name of the language that the video is primarily in. For example, if the speaker is speaking English, then the video is in English.

The language must exist on the site. If it doesn't exist, the API will waggle its finger at you.

Example: `"language": "English"`

state — Required. Possible values:

- 1 - live
- 2 - draft

Example: `"state": 1`

summary — Required. Short summary of the video formatted in Markdown. Should be no more than a single paragraph of a few sentences.

description Longer description of the video in Markdown. Outlines, linked timecodes, etc would go here.

tags List of tags.

If you pass in tags and they don't exist, the API will create them for you. If they do exist, the API will associate the video with the existing tag objects. (Yay!)

Example: `"tags": ["web", "django", "beard"]`

Note: If you're updating a video, you have to pass in the complete set of tags every time. If you pass no tags, it'll remove them assuming that you meant to remove all the tags.

---

speakers List of speaker names

If you pass in speaker names and they don't exist, the API will create them for you. If they do exist, the API will associate the video with the existing speaker objects. (Yay!)

Example: `"speakers": ["Carl Karsten", "Chris Webber"]`

Note: If you're updating a video, you have to pass in the complete set of speakers every time. If you pass no speakers, it'll remove them assuming that you meant to remove all the speakers.

---

source_url The url where the video resides. For example, if this video were hosted on YouTube, then you'd provide the YouTube url for it.

FIXME - Finish documenting fields. See code for the rest of the fields.

Here's minimal JSON example for a video:

```
{
  "category": "Test Category",
  "title": "Test video title",
  "language": "English",
  "state": 1
}
```

Here's a slightly longer one:

```
{
  "category": "Test Category",
  "title": "Test video title",
  "language": "English",
  "state": 1,
  "speakers": ["Jimmy Discotheque"],
  "tags": ["test", "bestever"],
  "summary": "Jimmy tests things out.",
  "description": "Tests\nAnd more tests."
}
```

CHAPTER 2

Part 2: Contributor's Guide

This guide is for current and future contributors. richard is a Free Software project and this guide covers everything you need to know about the project to help out.

2.1 Join this project!

Interested in building a video index system? Interested in running one? Then you should be interested in contributing to this project!

2.1.1 Project details

Code: https://github.com/pyvideo/richard

Issue tracker: https://github.com/pyvideo/richard/issues

Documentation: Documentation is in the `docs/` directory including docs for installing, upgrading, deploying, contributing, ...

http://richard.readthedocs.org/

IRC channel: `richard` on `irc.freenode.net`

I'm in Eastern Time (UTC -4 or UTC -5) and usually am on during the week during daylight hours.

2.1.2 Want to help?

I'd love to have help! Here are things I need help with:

- documentation work
  - verifying correctness of the existing documentation
  - fixing typos and errors in the documentation
  - fixing TODO items in the documentation
  - fixing flow issues – if you're trying to get something done, are the important bits in the same place?

  - fixing missing documentation – are there important things that aren't covered?
  - removing documentation – are there things that are complex enough that we should change richard so we don't have to explain those things?
- styling work
  - make the default site look better
  - fix pages so they work well on desktop browsers and mobile browsers
- code work
  - correct egregious errors which are probably due to misunderstandings of how Django and other components work
  - working on issues (bugs and new features) listed in the issue tracker at https://github.com/pyvideo/richard/issues
  - fixing TODO items in the code
- testing work
  - write tests – there's test infrastructure, but very few tests
  - test richard – there are likely issues I haven't discovered, yet
  - test richard on browsers other than Firefox nightly – I use Firefox nightly and haven't looked at richard on anything else
  - test richard on mobile devices – does it work well in tablets? does it work well on phones? what issues should get fixed?
- community work
  - do you know other groups looking for video index software? tell them about richard, help them set up richard, ...
- project infrastructure work
  - is there infrastructure missing in this project that would make it easier for you to contribute? if so, what?

    is there infrastructure missing in this project that would make it easier to set up on PaaS systems or other systems? if so, what?

"Gah! That's a huge list! It's too daunting!" That's ok! Hop on IRC and say hi and we can go from there!

2.2 Installing, running and testing richard

This covers how to clone richard and set it up for easy hacking.

Note: richard is still under heavy development. As such, the documentation for it is constantly in flux and probably outdated and the installation guide may have as much of a chance of helping you install richard as it does helping you make a quiche.

I'm really sorry about that. If you find issues, please let us know:

https://github.com/pyvideo/richard/issues

---

richard requires a bunch of stuff to run. I'm going to talk about this stuff in two groups:

1. stuff that you should install with your operating system's package manager
2. Python packages that you should install with pip in a virtual environment

2.2.1 Install things with your package manager

You need the following things all of which should be provided by your system/package manager:

- Python 2.7 or 3.3+
- pip
- virtualenv
- git

Python 2.7

Debian:

```
$ apt-get install \
      libxml2 \
      libxml2-dev \
      libxslt-dev \
      python-pip \
      python-virtualenv \
      libpython-dev \
      libz-dev
```

Fedora:

```
$ yum install \
      libxml2-devel \
      libxslt-devel \
      python-pip \
      python-virtualenv \
      python-devel \
      zlib-devel
```

Python 3.3+

FIXME: Please provide instructions.

2.2.2 Install and configure richard

Get the code, set up virtual environment and install requirements

First, you need the code. Clone the repository from github:

```
$ git clone git://github.com/pyvideo/richard.git
```

Create a virtual environment:

```
$ cd richard
$ virtualenv ./venv/
```

Note: If you want to use virtualenvwrapper or want to set things up differently, feel free to do so!

Make sure to activate the virtual environment every time you go to use richard things. You can do that like this:

```
$ . ./venv/bin/activate
```

Use pip in the virtual environment to update pip to the latest version:

```
$ pip install -U pip
```

Use pip in the virtual environment to install richard and the development requirements:

```
$ pip install -e ".[dev]"
```

(Optional) use postresql

If you want to also install with postgres support, you'll need to install postgresql and the bits you need to compile the postgresql driver.

On Debian:

```
$ apt-get install \
    postgresql \
    build-essential \
    libpq-dev
```

Then run in your virtual environment:

```
$ pip install -e ".[postgresql]"
```

Configure

You should be able to use the `Dev` configuration specified in `richard.config.settings`. This is the default used by `manage.py`.

The settings should work out of the box, but you can change them as you see fit.

(Optional) use postgres

Set the `DATABASE_URL` environment variable. See http://django-configurations.readthedocs.org/en/latest/values/#configurations.values.DatabaseURLValue for details.

Set up database schema

To set up the database schema and create the superuser, run:

```
$ ./manage.py migrate
```

Set up superuser account

To create a superuser account, run:

```
$ ./manage.py createsuperuser
```

The username and password don't matter–you'll never use them. However, the email address you use does since that needs to be the same as your Persona account.

All set!

You should have richard installed now. Any time you update the richard code, you'll want to install any requirements changes:

```
$ pip install -e ".[dev]"
```

and run migrations:

```
$ ./manage.py migrate
```

2.2.3 Run the tests

Richard uses pytest-django to discover tests.

Activate the virtual environment, then run the tests:

```
$ py.test tests
```

2.2.4 Run the server

Run the server like this:

```
$ ./manage.py runserver --traceback
```

Then point your browser at `http://localhost:8000/`.

2.2.5 Install the pre-commit hooks (optional)

richard uses pre-commit package to install various pre-commit hooks to lint the code when you create new commits. Install the hooks by running:

```
$ pre-commit install
```

The configuration of the hooks is done in `.pre-commit-config.yaml`. To ignore all the errors and proceed with the commit, use the ``--no-verify` option to the `git commit` command. To ignore specifc hooks, you can specify a comma-separated list of hook ids (available in `.pre-commit-config.yaml`) in the environment variable `SKIP`.

2.2.6 Set up sample data (optional)

You can add some sample data to your database which makes development a little easier since you can see what things look like. To do this, do:

```
$ ./manage.py generatedata
```

Note: This doesn't affect running tests at all. You can always delete sample data later.

FIXME: Running `generatedata` a second time will fail because slugs won't be unique.

2.2.7 Troubleshooting

I can't log in

First, make sure your administrator account has an email address associated with it. This is the email address you will log in with Persona.

After that, wee the django-browserid troubleshooting docs for more details.

2.3 Tests

2.3.1 Testing requirements

richard uses pytest-django to tie the Django test system to pytest.

We use FactoryBoy to generate model instances and test data.

2.3.2 Running tests

To run the tests, make sure your virtual environment is activated and then:

```
py.test tests/
```

You can see more options by doing:

```
py.test --help
```

Once you get a runline you like, put it in a bash script.

2.3.3 Add new tests

Locations

Conventions

Tests go in directories like `tests/test_<APPNAME>/`.

We use Factory Boy to generate model instances. Factory classes go in `tests/test_<APPNAME>/factories.py`.

Test modules should be named `test_*.py`.

Test classes should be named `Test*` and should extend Django's `UnitTest` class.

Functions and methods should be named `test_*`.

Use the non-camel-case versions of `assertXyz` and friends if they exist.

See existing tests for examples.

We're not shooting for 100% code coverage—only write tests that are compelling.

Make sure tests are documented and it's clear what's being tested and how.

2.4 Documentation

2.4.1 In the code

Documentation in the code is really helpful. Please add comments where you think it's necessary.

We like to use docstrings for classes, methods and functions. They should be in reStructuredText format. Something along these lines, though most of our docstrings aren't as formal or complete:

```
def foo(arg1, arg2):
    """Foo does something interesting

    :arg arg1: Controls whether or not to bar
    :arg arg2: Name of the baz to use

    :raises ValueError: If arg2 is not a valid baz.

    :returns: A bat.
    """
```

The purpose in-code documentation is three-fold:

1. to clarify complex code so it's easier to discern what it's doing
2. to make it clear why the code is doing what it's doing
3. to document any issues the code might have

2.4.2 The administrator's guide and contributor's guide

These two guides live in `docs/`.

We use Sphinx to structure and build our documentation.

To build the documentation, do:

```
$ cd docs/
$ make html
```

The docs are available in HTML and PDF forms at http://richard.readthedocs.org/. Whenever something is committed and pushed to the master branch, GitHub pings ReadTheDocs and the docs that are there are rebuilt.

2.5 Updating richard

If you're hacking on richard and running it from a git clone, then you'll need to do the following periodically.

2.5.1 Updating virtual environment

Make sure you've activated your virtual environment.

Then:

```
$ ./venv/bin/pip install -e ".[dev]"
```

Note: If you think you hosed your virtual environment, just wipe it and build a new one.

2.5.2 Perform migrations

richard uses south to manage database migrations.

To migrate your database to the latest schema, do:

```
$ ./manage.py syncdb
$ ./manage.py migrate
```

Note: If you're already up-to-date, then this won't do anything.

2.5.3 Update pre-commit hooks

richard uses pre-commit package to install various pre-commit hooks to lint the code. If there are any changes to the pre-commit configuration in .pre-commit-config.yaml, update the hooks by running:

```
$ pre-commit install
```

2.6 Project layout

When you do a `git clone ...` of richard, you end up with a directory tree like this:

```
richard              -- repository root
|
|- docs/             -- documentation
|- tests/            -- all the tests
\- src/
   \- richard/       -- richard django project
      |- base/       -- base code shared by the other apps
      |- config/     -- settings and configuration
      |- pages/      -- code for "about", "contac", etc pages
      |- sampledata/ -- code for loading sampledata from apps
      |- sitenews/   -- code for sitenews
      \- videos/     -- code for videos and search
```

Here's what's there:

docs/

Documentation for the project build wtih Sphinx and formatted in restructuredtext.

tests/

This is where all the tests go.

src/richard/

This is the "Django project" part of the project and where the "Django apps" go. There are a few:

base

This is where shared code for all the apps go as well as code central to the Django project. Middleware, context processors, base templates, static assets, etc.

config

This is where the settings and wsgi files live.

pages

Like django.contrib.flatpages except that nothing is in the database—it's all done with templates.

sampledata

Small utility app that will load sample data from other apps for development.

sitenews

Bare-bones "blog" for your site allowing you to specify site news like changes, new conferences added, etc.

videos

This app does most of the work and manages speakers, categories (conferences, user groups, etc), videos and search.

2.7 Conventions

2.7.1 Python code

- PEP-8: http://www.python.org/dev/peps/pep-0008/
- PEP-257: http://www.python.org/dev/peps/pep-0257/
- We use flake8 along with a few other pre-commit hooks to lint Python code.

pep8 covers Python code conventions.

pep257 covers Python docstring conventions.

Minor caveats:

- We use Sphinx, so do function definitions like they do: http://packages.python.org/an_example_pypi_project/sphinx.html#function definitions.
- Don't kill yourself over 80-character lines, but it is important.
- If you're flummoxed by the conventions, just send me the patch and as long as it functionally works, I can do a cleanup pass in a later commit.

2.7.2 HTML/Django templates

http://mozweb.readthedocs.org/en/latest/coding.html#html

We use the same HTML style conventions that Mozilla webdev does.

2.7.3 CSS

Todo

css conventions

2.7.4 JavaScript

http://mozweb.readthedocs.org/en/latest/js-style.html

We use the same JavaScript style conventions that Mozilla webdev does. We use jshint pre-commit hook to lint JavaScript code.

2.7.5 Git

I encourage good commit messages in a form that works well with git's various commands. Something like http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html. except that I don't care about verb tense or

capitalization and if the commit message is tied to a bug report, the bug report number should be the first thing in the first line. Here's the tbaggery example with some adjustments:

```
475. short summary (50 chars or less)

More detailed explanatory text, if necessary.  Wrap it to about
72 characters or so.  In some contexts, the first line is
treated as the subject of an email and the rest of the text as
the body.  The blank line separating the summary from the body
is critical (unless you omit the body entirely); tools like
rebase can get confused if you run the two together.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet,
  preceded by a single space, with blank lines in between, but
  conventions vary here

- Use a hanging indent
```

Why? Here's the reasons:

- 50 characters or less works well with the various git commands that show only the summary line and also on github.
- Having the bug number as the first thing makes it easy to see which commits covered which bugs without parsing the commit message. We do that a lot ("When did the fix for bug xyz land?").
- Wrapping the subsequent paragraphs allows them to show up nicely in git output as well as on github.

Why not the other things? Here's the reasons:

- Capitalization or non-capitalization for a phrase doesn't affect the output of git commands or my ability to quickly parse a summary.
- Ditto for verb tense.
- I'm all for ditching convention baggage for things that don't matter.

2.8 Resources

This chapter is a list of resources and notes that I thought would be helpful to new people who want to contribute to this software. There's also some design decisions and code conventions in here, too.

2.8.1 Project scaffolding

Settings

richard has migrated to django-configurations which makes settings in Django behave more like regular old class inheritance. The classes `Base`, `Testing`, `Dev` and `Prod` in the `settings.py` module all handle the expected domains. You can override these by:

- creating a `settings_local.py` file
- `from . import settings`
- creating your class (or creating a Dev / Prod class) and extending from `settings.Base`
- Updating your deployment scripts to run `manage.py --settings richard.config.settings_local`

Requirements / environments / deployment

- virtualenv and pip

  richard uses virtualenv and pip to build the environment for richard to run.

  Requirements are listed in the `setup.py` with `extra_requires` defined for local development and postgres. Since richard is meant for deployment rather than a framework, requirements are pinned hard. This makes it easy to install various options without having multiple requirements files.

Documentation

- Sphinx and restructuredtext

  Documentation is in `docs/` and uses Sphinx for organizing and building it and restructuredtext for the markup.

2.8.2 Tools

- git, github and ProGit

  richard uses git for version control. This has a big effect on how the project evolves in respects to code changes.

- pyflakes

  pyflakes is a great code checker that eliminates a class of possible errors from your code. I highly recommend using it.

  I use it with Emacs. This page covers setting up pyflakes with Emacs in a couple of different ways.

  Another way to run it is as a pre-commit hook with check.py.

2.8.3 Django / py.test / haystack / whoosh / django-rest-framework

- Django

  This software is built using Django. I tried to use Django pieces where possible.

- pytest-django and pytest

  We use pytest and the django wrapper.

- django-haystack and whoosh

  This runs the search system. I picked whoosh because it's a pure Python package and thus really easy to install and use. That makes richard easy for contributors to get up and running.

  You can pick a different backend by setting the appropriate configuration in `settings_local.py`. See the django-haystack documentation for details.

- south

  South manages schema and data migrations independent on the database.

- django-rest-framework

  django-rest-framework provides a RESTful API that can be used to retrieve videos programmatically and can ease importing many videos for site admins.

2.8.4 HtML / CSS / JavaScript

- Learning HTML at MDN, Learning CSS at MDN and Learning JavaScript at MDN

  These are great references for learning HTML, CSS and JavaScript. Highly recommended reading before you jump into the user-interface related code.

2.8.5 Video and Universal Subtitles

- Using HTML5 audio and video covers HTML5 video tag.
- Universal Subtitles and Universal Subtitles wiki

  This is the subtitling system we're using. Their wiki covers embedding, wrapping, and the API.

2.9 Authors

Many thanks to all of these people who have contributed code to this project. Without them, this project would be nothing.

- Will Kahn-Greene
- David Winterbottom
- Reiner Gerecke
- Alejandro Gómez
- Ricky Rosario
- Adam Zapletal
- Nes.quik
- Pavel Savchenko
- Spencer Herzberg
- Frank Ploss
- Riley Labrecque
- Eka Putra
- Eduard-Cristian Stefan

- Martin Bächtold
- Sheila Miguez
- Trey Hunner
- Paul Collins
- Benjamin Bertrand
- Burak Guven
- Wes Turner
- Magnun Leno

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

Documentation todo

Todo

flesh out how to override css, js and templates

(The original entry is located in /var/build/user_builds/richard/checkouts/latest/docs/admin/administration.rst, line 20.)

Todo

css conventions

(The original entry is located in /var/build/user_builds/richard/checkouts/latest/docs/contributors/dev_conventions.rst, line 42.)